UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

Trabajo Fin de Máster

Detección de vehículos utilizando Velodyne para aplicaciones de navegación autónoma

Alumno: D. Alberto Carpintero Sanz

Director: D. Miguel Ángel Sotelo Vázquez

Tribunal:

Presidente: D. David Fernández Llorca

Vocal 1º: D. Pedro Gil Jiménez

Vocal 2º: D. Miguel Ángel Sotelo Vázquez

Calificación:…….

Fecha:……**...**

A mis padres, Jesús y Charo,
a mis tíos, Javi y Rosa,
a Eduardo, Estrella y Edu,
y a Leire.

Agradecimientos

A Miguel Ángel Sotelo, por haberme dado la oportunidad de realizar este TFM, así como de haberme prestado todo su conocimiento acerca de este fantástico mundo.

A Rubén Izquierdo, quien durante todo el desarrollo de este TFM ha estado apoyando, aconsejado y resolviendo todas dudas.

A mis padres, por su esfuerzo, lucha y constancia por mi educación y bienestar.

A Leire, por ser uno de los pilares en mi vida, gracias por apoyarme y ayudarme a conseguir todo lo que me propongo.

A mi amigo, Sergio por todas las tardes de estudio y comprensión para entender las cosas.

Muchas gracias a todos.

Índice general

Índice de figuras

Parte I

Resumen

0.1. Resumen

El objetivo fundamental de este trabajo es proporcionar una solución a uno de los problemas existentes en los sistemas de navegación autónoma, cómo es problema de la detección de vehículos en el entorno.

Para ello, se ha desarrollado un algoritmo capaz de detectar los vehículos existentes en el entorno mediante la información tridimensional proveniente de un sistema de medición láser ubicado en la parte superior del vehículo. Esta información es empleada para desarrollar una segmentación del entorno para determinar los obstáculos del mismo, aplicación de un modelo para determinar la profundidad, el ancho y el ángulo de los vehículos y una integración temporal de los resultados que permita realizar el seguimiento de los vehículos detectados entre *frames*.

Palabras clave

- Navegación autónoma.
- Detección de vehículos.
- Escáneres láser.
- Nube de puntos.

0.2. Abstract

The main goal of this work is to provide a solution to one of the problems in autonomous navigation systems, such as the detection of vehicles in the environments.

For this purpose, an algorithm has been developed capable of detecting the existing vehicles in the environments through the three-dimensional information coming from a laser measuring system located in the upper part of the vehicle. This information is used to develop a segmentation of the environments to determine the obstacles, application of a model to determine the depth, width and angle of the vehicles and a temporal integration of the results that allows to track the detected vehicles between frames.

Palabras clave

- Autonomous navigation.
- Vehicle detection.
- Laser scanners.
- Point cloud.

Parte II

Memoria

Capítulo 1

Introducción

En los últimos años el sector del automóvil ha incluido numerosos sistemas de ayuda a la conducción en sus vehículos. Comenzó con el aviso por cambio involuntario de carril, continuó con los sistemas de aparcamiento guiados y hoy en día se dispone desde sistemas que logran aparcar por sí mismos hasta sistemas que realizan un frenado del vehículo en caso de posible colisión.

Actualmente se están desarrollando multitud de proyectos de investigación dedicados ya no solo a la constitución de sistemas de ayuda a la conducción si no a la obtención de sistemas autónomos de transporte. El trabajo desarrollado tanto por parte de los fabricantes como de empresas privadas o instituciones públicas está dando como resultado sistemas cada vez más complejos, útiles y fiables.

Este trabajo de fin de máster se inscribe dentro una serie proyectos dedicados a la consecución de sistemas de asistencia a la conducción y pretende ser tanto un sistema con aplicación independiente, como un elemento que permita la mejora de los ya existentes.

En este trabajo se desarrolla un algoritmo que es capaz de detectar los vehículos existentes en el entorno proporcionando la profundidad, el ancho y el ángulo de cada uno de ellos. La detección de vehículos permitirá llevar a cabo un control autónomo del vehículo en cualquier situación.

Para este proyecto se usará una base de datos generada a través del LiDAR Velodyne HDL-32E que incorpora el coche autónomo de la Universidad de Alcalá. Esta base de datos consta de múltiples secuencias grabadas tanto en entornos urbanos como extraurbanos.

El coche autónomo de la Universidad de Alcála tiene además un sistema de visión estéreo en color, un GPS, dos radares de corto alcance, un láser de barrido horizontal Sick y todos los datos recogidos por el vehículo a través del bus CAN (*Controller Area Network*) relativos a la conducción. Esto le permite realizar una conducción autónoma gracias control total sobre el volante y los pedales del vehículo.

Figura 1.1: Coche autónomo de la Universidad de Alcalá.

1.1. Estructura

El documento que describe el algoritmo se divide en los siguientes capítulos:

- **Capítulo 1**: Introducción: Se plantean los términos en los que se desarrolla el trabajo además de la estructura del documento.

- **Capítulo 2**: Estado del arte: Se realiza un breve resumen de las distintas aportaciones o técnicas llevadas a cabo por distintos grupos de investigación en aplicaciones similares a la planteada en este trabajo, además se enumeran los objetivos que se persiguen con el desarrollo de este TFM.

- **Capítulo 3**: Plano principal del entorno: Se describe cómo se obtiene la nube de puntos y cómo a partir de esta nube se obtiene el plano principal del entorno para posteriormente ajustar este plano al plano OXY.

- **Capítulo 4:** Segmentación de la nube para detectar obstáculos: Se expone el algoritmo utilizado para realizar la detección de obstáculos en el entorno y cómo extraer nubes de puntos independientes para cada obstáculo detectado.

- **Capítulo 5:** Aplicación de un modelo para determinar el ancho, profundidad y ángulo de los vehículos: Se describe cómo determinar si un obstáculo es un vehículo o no y se procederá a aplicar un modelo para determinar su ancho, profundidad y ángulo.

- **Capítulo 6:** Integración temporal de los resultados: Se expone como relacionar los diferentes vehículos entre *frames* con el objetivo de mejorar el modelo del vehículo a lo largo de los *frames.*

- **Capítulo 7:** Resultados: Se explican los resultados obtenidos con el sistema después del procesamiento de varias nubes de puntos.

- **Capítulo 8:** Conclusiones y Trabajos Futuros: Se resumen las conclusiones obtenidas durante la realización de proyecto y se proponen ciertas mejoras que pueden plantearse y desarrollarse.

Capítulo 2

Estado del Arte

2.1. Introducción

Durante los últimos años han aparecido multitud de sistemas de ayuda a la conducción o ADAS (*Advanced Driver Assistence System*) que han permitido aumentar la seguridad de los vehículos y prevenir los accidentes. Así como sistemas ITS (*Intelligent Transportation System*) que permiten mejorar la gestión del tráfico, disminuir las emisiones de gases contaminantes y reducir el consumo de combustibles.

Establecer cuál es el espacio libre de circulación del vehículo es de suma importancia para los ADAS. Con él se puede determinar cuándo un vehículo va a abandonar, o no, la zona por la que puede transitar y actuando en consecuencia para evitar accidentes potenciales.

El espacio libre de circulación del vehículo no solo es útil para evitar accidentes y mejorar la seguridad de los vehículos, sino también para los procesos de conducción autónoma de vehículos.

Detectar los límites del espacio libre de circulación del vehículo en escenas urbanas es sumamente complicado debido a las variadas y complejas situaciones que se dan en estos entornos.

El desarrollo de nuevos sistemas que permitan detectar el espacio libre en escenas urbanas es fundamental para el objetivo de conseguir vehículos autónomos y mejorar los sistemas ADAS existentes.

2.2. Comienzos

Existen tres razones que explican la paulatina evolución en los últimos años de los sistemas enfocados a la obtención de vehículos inteligentes en todo el mundo:

- La necesidad de reducir la mortandad en las carreteras y los gastos socioeconómicos que ello provoca.
- La tecnología que se ha desarrollado por parte de los investigadores en visión computacional.
- El crecimiento exponencial de la velocidad de los procesadores que permiten ejecutar en tiempo real algoritmos más complejos de procesamiento de imágenes.

Un proyecto pionero fue PROMETHEUS (*Program for European Traffic with Highest Efficiency and Unprecedented Safety*) [1], desarrollado entre los años 1986 y 1994 como parte del programa EUREKA (*European Research Coordination Agency*) cuyos participantes fueron algunos institutos de investigación, empresas de electrónica y fabricantes de vehículos de 19 países europeos. Los principales objetivos del programa iban enfocados a cuatro áreas de trabajo: mejora del tráfico y la circulación en las carreteras, seguridad en la conducción y reducción del número de accidentes, gestión eficiente de las flotas de transporte de mercancías y reducción de las emisiones contaminantes. Dentro del programa PROMETHEUS hubo siete subprogramas:

- Sistemas a bordo para monitorización de vehículos y asistencia al conductor.
- Comunicación entre vehículos.
- Comunicación carretera-vehículo para control de tráfico.
- Desarrollo de los componentes microelectrónicos requeridos.
- Uso de inteligencia artificial y desarrollo de software.
- Comunicación dentro del sistema del vehículo y el conductor, vehículo y vehículo, además de comunicaciones para los vehículos de la red vial en general.
- Efectos del cambio de vehículos en el entorno de tráfico.

Como resultado del proyecto PROMETHEUS se crearon diversos prototipos o sistemas, entre los que destacan: VaMoRs, Vamp, MOB-LAB, VITA (*Vision Technology Application*) [2], ARGO [3] o GOLD (*Generic Obstacle and Lane Detection*) [4].

Otro proyecto revolucionario en Europa fue DRIVE (*Dedicated Road Infrastructure for Vehicle Safety in Europe*) [5], lanzado en el año 1989 bajo el control de la CEC (*Commission of European Communities*) y cuyo objetivo era mejorar las infraestructuras, la seguridad y la eficiencia de las carreteras. En 1992 se lanzó DRIVE II, pensado para la implementación de los proyectos pilotos que habían sido desarrollados en la fase anterior. Dentro de los programas DRIVE y DRIVE II surgieron muchos subprogramas dentro de las siguientes áreas:

- Administración y mejora del transporte público.
- Administración de los transportes de carga y flotas.
- Asistencia a los conductores y manejo cooperativo.
- Administración del tráfico interurbano.
- Administración del tráfico urbano.
- Información de tráfico y de viaje.
- Administración de la demanda.

Un proyecto de gran relevancia dentro de DRIVE fue SOCRATES (*System of Cellular Radio for Traffic Efficiency and Safety*) [6], cuyo objetivo fue utilizar ondas de telefonía para intercambiar información entre computadores de a bordo en los vehículos. Ofrecía los siguientes servicios:

- Guía dinámica de ruta: daba recomendaciones al conductor de rutas basadas en la información de la carretera, así como condiciones actuales y previstas del tráfico.
- Administración de vehículos de transporte: permitía monitorizar la posición y la situación de los vehículos.
- Administración de transporte público y sistemas de información: organizaba horarios flexibles de transporte y ofrecía información a los usuarios.
- Advertencias de peligro: los sistemas de comunicación a bordo de los vehículos daban a los conductores advertencias de peligros, tanto de accidentes como de condiciones climatológicas adversas.
- Información al conductor: ofrecía la capacidad de dar a los conductores información sobre los puntos problemáticos y sobre las condiciones de tráfico.
- Información para la gestión de tráfico: permitía reunir diferentes tipos de datos sobre el tráfico.

Aunque los programas europeos tienen un gran renombre, en Japón aparecieron importantes proyectos a comienzos de la década de los 90, participando empresas como Nissan, Fujitsu o MITI. Se desarrolló el sistema PVS (*Personal Vehicle System*) [7], cuyo propósito era la creación de un vehículo capaz de detectar las líneas blancas y los obstáculos en la carretera mientras se movía de forma autónoma basándose en esa información. En 1996, la AHSRA (*Advanced Cruise- Assist Highway System Research Association*) fue creada por algunas empresas automovilísticas y centros de investigación para implementar sistemas de seguridad y mejora de infraestructuras.

En EEUU se creó en 1995 el NAHSC (*National Automated Highway System Consortium*) y en 1997 la IVI (*Intelligent Vehicle Initiative*), dando como resultado una gran cantidad de prototipos y sistemas. Dos proyectos importantes que iniciaron el campo de los vehículos inteligentes en el país fue ADVANCE (*Advance Driver and Vehicle Advisory Navigation Concept*) y Mobility 2000. El primero tenía objetivos como: mejorar la movilidad de los viajeros, reducir los tiempos de viaje y los costes de operación, minimizar los costes de la infraestructura de transporte, aumentar la seguridad en las autopistas, disminuir el consumo de energía en los transportes y reducir la contaminación y el ruido provocado por los mismos. El segundo fue desarrollado por algunas industrias, el Gobierno y ciertas universidades, enfocándose en la resolución de problemas de tráfico urbano, mejora de la seguridad, la productividad y la eficiencia del uso de la energía. Uno de los grupos de investigación más importantes en este área es Navlab (*Navigator Laboratory*) de la Universidad de Carnegie Mellon, donde se han
fabricado once vehículos y sus proyectos se enmarcan dentro de líneas de investigación como: autopistas automatizadas, prevención de colisiones o asistencia al conductor. En 2004, DARPA (*US Defense Advanced Reseach Projects Agency*) comenzó a organizar el

Grand Challenge, un concurso a nivel mundial donde quince vehículos totalmente autónomos (ver figura 2.1) deben navegar por diferentes entornos que varían cada año de competición.

Figura 2.1: Vehículo autónomo desarrollado para DARPA Urban Challenge 2007.

2.3. Métodos de detección de vehículos

La detección vehículos en el entorno de circulación se basada en la información 3D del entorno y, por tanto, necesita sistemas capaces de realizar mediciones precisas de la información 3D, es por ello por lo que el principal sensor utilizado es el láser.

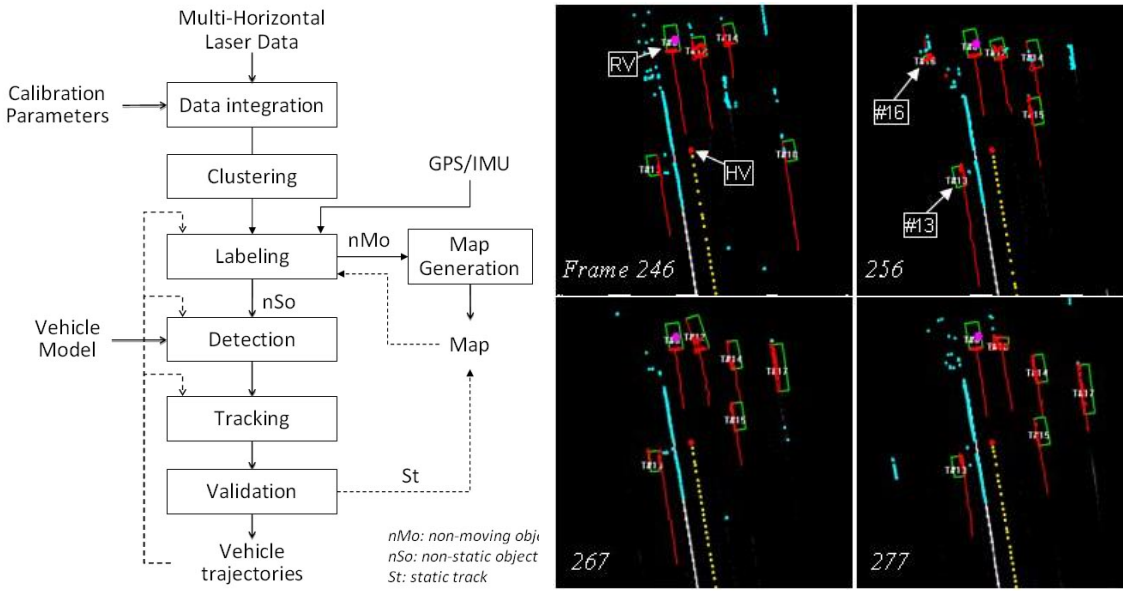
2.3.1. Basados en laser

Un sistema de medición láser consiste en uno o varios haces láser, fijos o móviles, que realizan mediciones en la trayectoria del haz láser. Los más empleados son los que cuentan con varios haces láser móviles, los cuales rotan entorno a un eje y proporcionan barridos de medidas.

- En [8] se presenta un algoritmo de detección y seguimiento de los vehículos del entorno utilizando un láser *Sick LMS291* y tres láser *Hokuyo UTM-30LX* .

  Para realizar la detección y seguimiento de los vehículos realiza el siguiente diagrama de flujo de procesos (ver figura 2.2 (a)):

1. Integración de datos: Asociación de datos de medidas simultáneas de múltiples sensores en diferentes puntos de vista.
2. *Clustering* para detectar los diferentes obstáculos del entorno.
3. Etiquetado inicial para discernir si un objeto es móvil, inmóvil o no es seguro. Este etiquetado se realiza en base al conocimiento a priori de los *frames* anteriores.
4. Generación de un mapa rejilla en el que cada celda representa la probabilidad de que exista un obstáculo en esa rejilla.
5. Detección de vehículos aplicando un modelo de vehículo sobre los *clusters* móviles o no seguros detectados. Este modelo se basa en un rectángulo.
6. Seguimiento de vehículos a lo largo de los frames.
7. Validación de las trayectorias y vehículos detectados.



(a) Diagrama de flujo de proceso (b) Resultados

Figura 2.2: Omni-directional detection and tracking of on-road vehicles using multiple horizontal laser scanners.

- En [9] se presenta un algoritmo para detectar los vehículos del entorno aplicando un modelo de vehículo en forma de L. Este método no se basa en la información de la secuencia del escaneo láser y, por lo tanto, soporta la fusión de sensores de múltiples láser.

  El algoritmo supone que la nube de puntos de los vehículos detectados tiene forma de L, tal y cómo puede observar en los puntos de color negro de la figura 2.3. El objetivo de este algoritmo es conocer el ancho, la profundidad y el ángulo de los vehículos, para ello busca el rectángulo que mejor se ajuste a los puntos de la nube. Este ajuste se realiza aplicando tres técnicas distintas: minimización del área (rectángulo verde de la figura 2.3), maximización de proximidad (rectángulo rojo de la figura 2.3) y minimización de la varianza (rectángulo azul

de la figura 2.3). Posteriormente, estas técnicas se comparan desde el punto de vista de la eficiencia computacional y ajuste del rectángulo para determinar pros y contras de cada técnica.

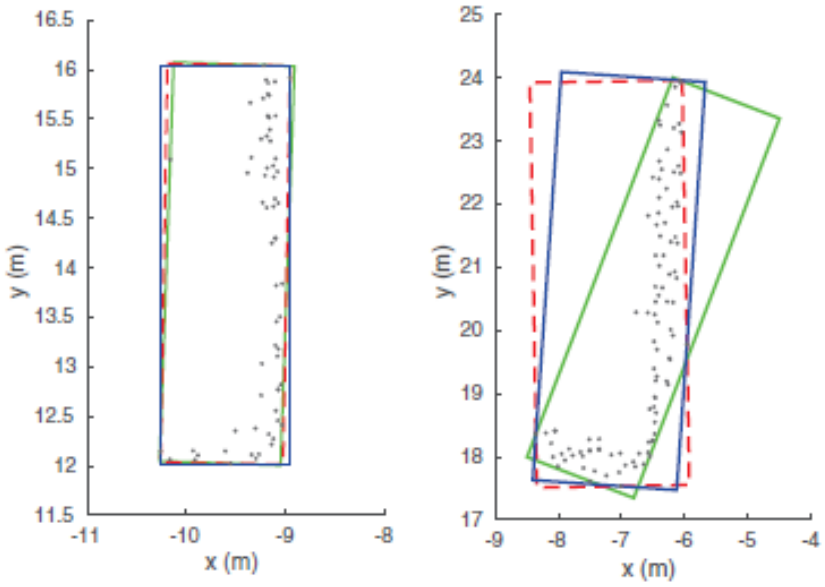


Figura 2.3: Efficient L-Shape fitting for vehicle detection using laser scanners.

2.4. Objetivos

El principal objetivo de este TFM es desarrollar un algoritmo capaz de detectar los vehículos existentes en cualquier tipo de entorno proporcionando la profundidad, el ancho y el ángulo de cada uno de estos, empleando para ello un sistema láser. Para lograr realizar el algoritmo final es necesario implementar una serie de algoritmos secundarios.

Los objetivos secundarios planteados para lograr el algoritmo final son los siguientes:

1. Obtención del plano principal del entorno.
2. Rotación de la nube de puntos para que esta quede horizontal con el plano principal.
3. Segmentación de la nube de puntos para discernir qué puntos de esta son obstáculos y cuáles de los obstáculos son vehículos.
4. Aplicación de un modelo para determinar la profundidad, el ancho y el ángulo de los vehículos.
5. Integración temporal de los resultados para realizar el seguimiento de los vehículos entre *frames*.

Capítulo 3

Plano principal del entorno

3.1. Obtención de la nube de puntos

La nube de puntos se ha obtenido de un sensor LiDAR, este es un dispositivo que permite determinar la distancia desde un emisor láser a un objeto o superficie utilizando un haz láser pulsado. Este láser se refleja en un espejo fijo y este a su vez en uno rotatorio lo que permite cubrir ángulos de 360º en horizontal (ver figura 3.1). La distancia al objeto se determina midiendo el tiempo de retraso entre la emisión del pulso y su detección a través de la señal reflejada.

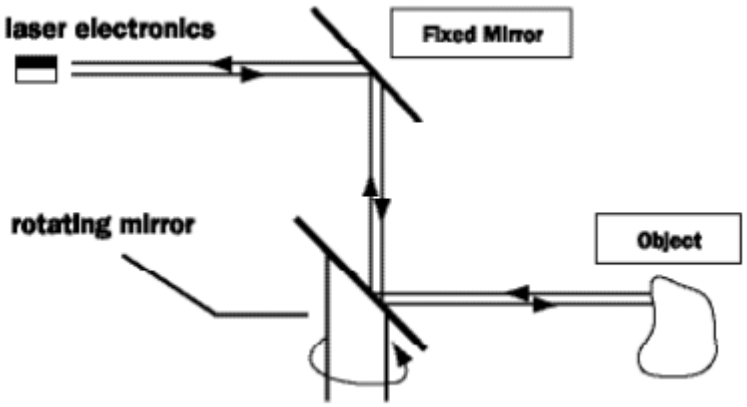


Figura 3.1: Ejemplo de funcionamiento de un LiDAR

En el caso del LiDAR de la marca Velodyne HDL-32E que se ha utilizado en este trabajo permite cubrir un área horizontal de 360º con una resolución de 0,5 grados y un área vertical desde +10,67 grados hasta -30,67 grados (41,33 grados) con una resolución de 1,3 grados gracias a los 32 haces, su velocidad de rotación es de 10 Hz y por lo tanto, proporciona una nube de puntos cada 100 ms.

La nube de puntos proporcionada por el sensor LiDAR está compuesta de puntos del entorno con coordenadas (x, y, z) en el sistema de coordenadas que se puede observar en la figura 3.2.

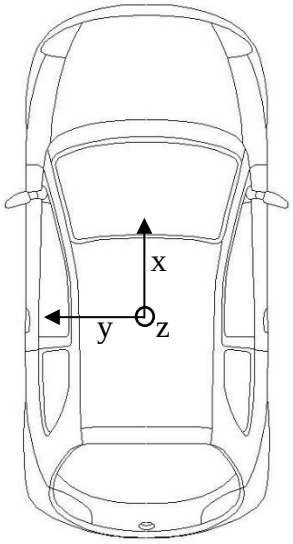


Figura 3.2: Sistema de coordenadas del LiDAR

3.2. Obtención del plano principal

3.2.1. RANSAC

RANSAC es un algoritmo iterativo utilizado para estimar los parámetros de un modelo matemático de conjunto de datos que contiene *outliers*. Otros métodos para estimar parámetros de modelos como mínimos cuadrados le dan el mismo peso a todos los datos, por lo tanto, la presencia de un *outlier* puede llegar a distorsionar el modelo obtenido.

La idea del algoritmo es utilizar la menor cantidad de puntos posibles para estimar el modelo y luego ver cuántos datos se ajustan al modelo estimado.

El algoritmo RANSAC es el siguiente:

1. Dado un modelo que requiere un mínimo de *n* para determinar sus parámetros, y un conjunto de datos *P* tal que el número de puntos en *P* es mayor que *n*, se sortea un subconjunto *S1* de *n* puntos de *P* para instanciar el modelo. Con el modelo instanciado *M1* se determina el subconjunto de decisión *S1** de puntos de *P* que están a menos de una distancia *t* de *M1*.
2. Si la cantidad de puntos en *S1** es mayor que un umbral *T* entonces se elige el subconjunto de decisión *S1** para computar el nuevo modelo *M1**

3. Si la cantidad de puntos en *S1** es menor que *T*, se sortea un nuevo subconjunto *S2* y se repite el proceso. Si luego de una cantidad de *N* número de pruebas no se obtiene un subconjunto de decisión que cumple con el umbral *T*, se resuelve el modelo con el subconjunto de decisión más grande obtenido, o se termina sin devolver modelo.

Como se puede ver en la explicación del algoritmo hay tres parámetros que estimar: (A) la distancia *t* para considerar si un punto es compatible o no con el modelo, (B) el número de iteraciones para buscar subconjuntos de decisión, y (C) el umbral *T* que es el número mínimo de puntos compatibles para considerar una buena estimación del modelo.

A: Umbral de distancia t para determinar compatibilidad dato/modelo

Se tiene que elegir un umbral *t* tal que la probabilidad de que un punto pertenece a un *inlier* es α. Esto requiere conocer la distribución de probabilidad de la distancia de un *inlier* del modelo. En la práctica se asume que el error es gaussiano de media cero y varianza σ.

B: Número de iteraciones

El número de iteraciones *N* se calcula en función de la probabilidad *p* de que por lo menos un subconjunto que se elige aleatoriamente está libre de *outliers*. ω es la probabilidad de que un punto dentro del conjunto de datos sea un *inlier*, por lo tanto,

$$\in = 1 - \omega \tag{3.1}$$

es la probabilidad de encontrar un *outlier*. Para que se cumpla la probabilidad *p* se tiene que cumplir que,

$$(1 - \omega^s)^N = 1 - p \tag{3.2}$$

Por lo tanto,

$$N = \frac{\log(1-p)}{\log(1-(1-\in)^s} \tag{3.3}$$

C: Número mínimo *T* de puntos en el subconjunto de decisión.

El número *T* se busca que sea similar al número de *inliers* que se supone que hay en el conjunto de datos. Por lo tanto, se toma:

$$T = (1-\epsilon)\, n \tag{3.4}$$

donde, *n* es la cantidad de puntos del conjunto de datos.

3.2.2. Plano principal del entorno

El plano principal del entorno es el correspondiente con la superficie de la carretera, este plano viene determinado por la siguiente ecuación.

$$ax + by + cz + d = 0 \tag{3.5}$$

Variando los parámetros $[a, b, c, d]$ es posible obtener superficies muy diversas como las que podemos encontrar en la figura 3.3.

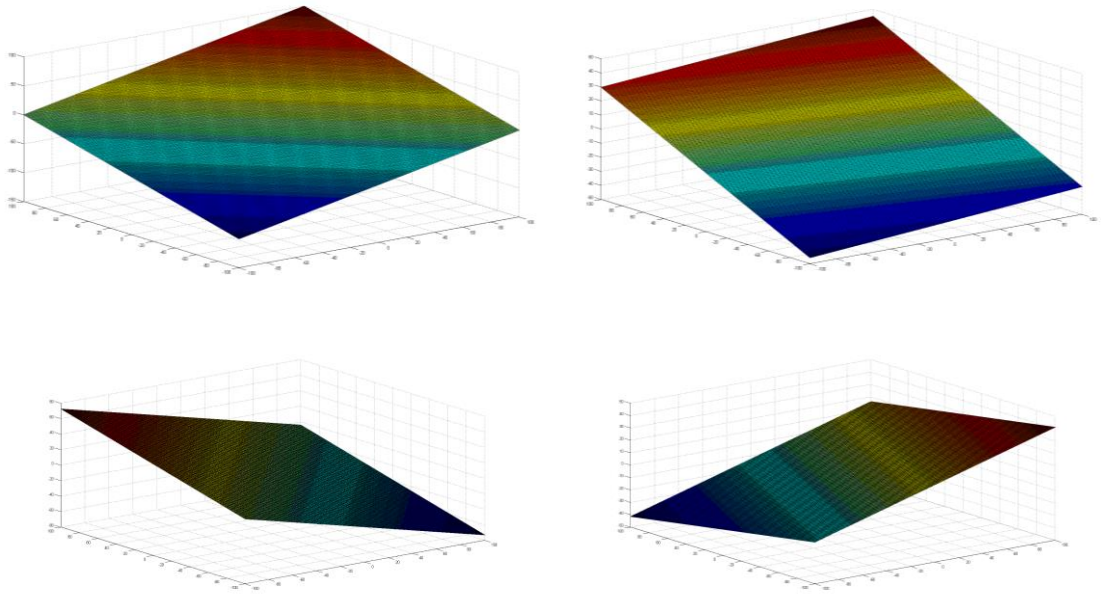


Figura 3.3: Superficies posibles con la ecuación 3.5

Para obtener los parámetros $[a, b, c, d]$ se ha aplicado el algoritmo RANSAC descrito en el apartado 3.2.1, este algoritmo proporciona los parámetros descritos anteriormente y los puntos de la nube que pertenecen a el plano descrito por esos parámetros (*inliers)* y el resto de puntos de la nube que no pertenecen a ese plano (*outliers).*

3.3. Ajuste del plano principal del entorno al plano OXY

3.3.1. Vector normal al plano

El vector normal a una superficie es un vector que es perpendicular a la superficie en un punto dado. Por lo tanto, dada la ecuación general del plano,

$$ax + by + cz + d = 0 \tag{3.6}$$

Se definen las componentes del vector normal unitario como,

$$n_x = \frac{a}{\sqrt{a^2 + b^2 + c^2}} \tag{3.7}$$

$$n_y = \frac{b}{\sqrt{a^2 + b^2 + c^2}} \tag{3.8}$$

$$n_z = \frac{c}{\sqrt{a^2 + b^2 + c^2}} \tag{3.9}$$

y la constante,

$$p = \frac{d}{\sqrt{a^2 + b^2 + c^2}} \tag{3.10}$$

Entonces la forma normal del plano es:

$$\hat{n}x = -p \tag{3.11}$$

donde, p es la distancia del plano al origen, cuyo signo determina el lado del plano en el que se encuentra el origen y $\hat{n}$ son las componentes del vector normal unitario dadas como,

$$\hat{n} = (n_x, n_y, n_z) \tag{3.12}$$

3.3.2. Matriz de rotación de un vector sobre otro vector

Se quiere obtener la matriz de rotación R que permite rotar el vector unitario a sobre el vector unitario b.

La matriz de rotación R viene determinada por:

$$R = I + [v]_x + [v]_x^2 \frac{1 - c}{s^2} \tag{3.13}$$

donde, I es la matriz identidad,

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.14}$$

v es el producto vectorial de los dos vectores a y b,

$$v = a \times b \tag{3.15}$$

$[v]_x$ es la matriz antisimétrica del producto cruzado de v,

$$[v]_x = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix} \tag{3.16}$$

s es la norma de v

$$s = \|v\| \tag{3.17}$$

y, c es el producto escalar de los dos vectores a y b,

$$s = a.b \tag{3.18}$$

3.3.3. Ajuste del plano principal del entorno al plano OXY

El objetivo es rotar el plano principal del entorno calculado en base a la nube de puntos obtenida para que este coincida con el plano OXY (ver figura 3.4)

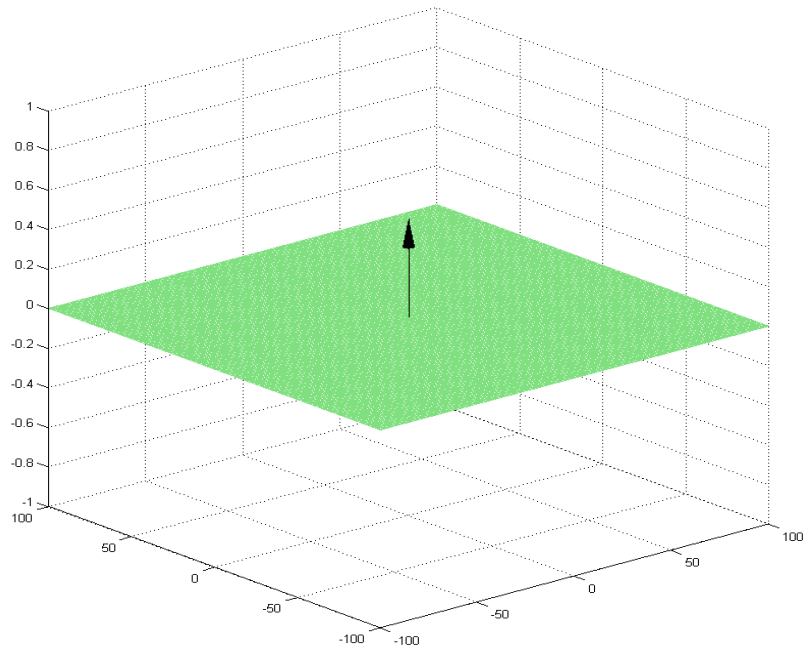


Figura 3.4: Plano OXY (plano de color verde) y vector normal a este plano (flecha de color negro)

El plano principal del entorno viene determinado por los parámetros $[a, b, c, d]$ que se han calculado en el apartado anterior 3.2.2.

El primer paso es calcular el vector normal al plano principal del entorno. Este cálculo se ha explicado en el punto 3.3.1. por lo tanto, sólo es necesario sustituir los parámetros $[a, b, c, d]$ del plano principal en la ecuación 3.6. para determinar el vector normal unitario.

El vector normal unitario del plano principal viene determinado tal y cómo se determina en la ecuación 3.12 por,

$$\hat{n}_p = (n_{xp}, n_{yp}, n_{zp}) \tag{3.19}$$

y la distancia del plano principal al origen p_p viene determinada por la ecuación 3.10.

Las coordenadas del vector normal unitario del plano OXY se expresan cómo,

$$\hat{n}_{xOXY} = 0 \tag{3.19}$$

$$\hat{n}_{yOXY} = 0 \tag{3.20}$$

$$\hat{n}_{zOXY} = 1 \tag{3.21}$$

Por lo tanto, el vector normal unitario se determina de la siguiente manera,

$$\hat{n}_{OXY} = \left(\hat{n}_{xOXY}, \hat{n}_{yOXY}, \hat{n}_{zOXY}\right) = (0,0,1) \tag{3.22}$$

Para transformar el plano principal del entorno en el plano OXY es necesario realizar una translación y una rotación del plano principal.

La rotación se calcula tal y cómo se explica en el apartado 3.3.2., sólo es necesario sustituir el vector a por $\hat{n}_p$ y el vector b por $\hat{n}_{OXY}$. El resultado devuelto por la ecuación 3.13 nos determinara la rotación R_p del vector $\hat{n}_p$ sobre $\hat{n}_{OXY}$.

La translación del plano principal con respecto al plano OXY se expresa de la siguiente manera,

$$T_p = p_p \hat{n}_{OXY} \tag{3.23}$$

Por tanto,

$$T_p = p_p \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.23}$$

Finalmente se obtiene que la translación es,

$$T_p = \begin{bmatrix} 0 \\ 0 \\ p_p \end{bmatrix} \tag{3.24}$$

Una vez se tiene la rotación y la translación del plano principal respecto al plano OXY, es necesario realizar la rotación y la translación a todos y cada uno de los puntos de la nube que tal y cómo se ha comentado en el apartado 3.1, estos tienen coordenadas (x, y, z) que hay que transformar. La transformación de los puntos viene determinada por la siguiente ecuación:

$$\begin{bmatrix} x_i' \\ y_i' \\ z_i' \end{bmatrix} = R_p T_p \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \tag{3.25}$$

donde, i es cada punto de la nube, (x_i', y_i', z_i') es el punto de la nube transformado, R_p es la matriz de rotación 3x3, T_p es la matriz de translación 3x1 determinada por la ecuación 3.24 y (x_i, y_i, z_i) es el punto de la nube sin transformar.

3.4. Resultados

En la figura 3.5 se puede observar la nube de obtenida a través del láser Velodyne tal y cómo se ha explicado en el apartado 3.1.

En la figura 3.6 se puede apreciar la nube de puntos obtenida a través del láser Velodyne (puntos de color verde). Los puntos de color rojo corresponden con los puntos de la nube anterior que pertenecen al plano principal. Los puntos mostrados son los *inliers* que determina el algoritmo de RANSAC explicado en el apartado 3.2.

En la figura 3.7 se puede observar la nube de puntos obtenida a través del láser Velodyne (puntos de color verde) y la nube de puntos ajustada al plano OXY (puntos de color azul), esta transformación se detalla en el apartado 3.3

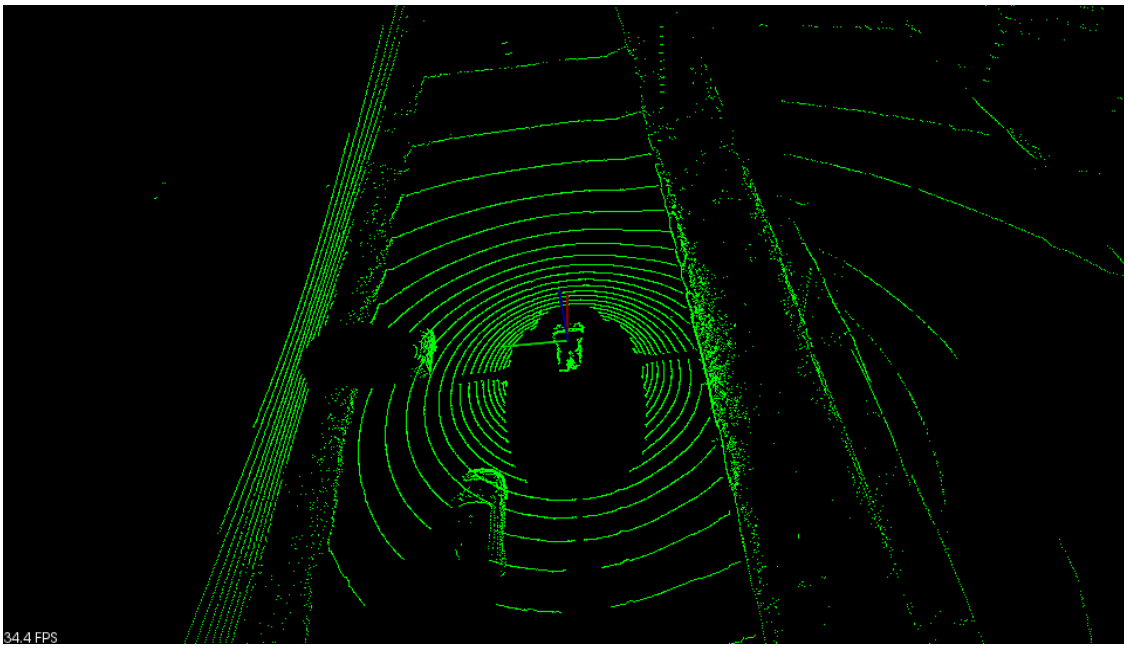


Figura 3.5: Nube de puntos obtenida a través del láser Velodyne.

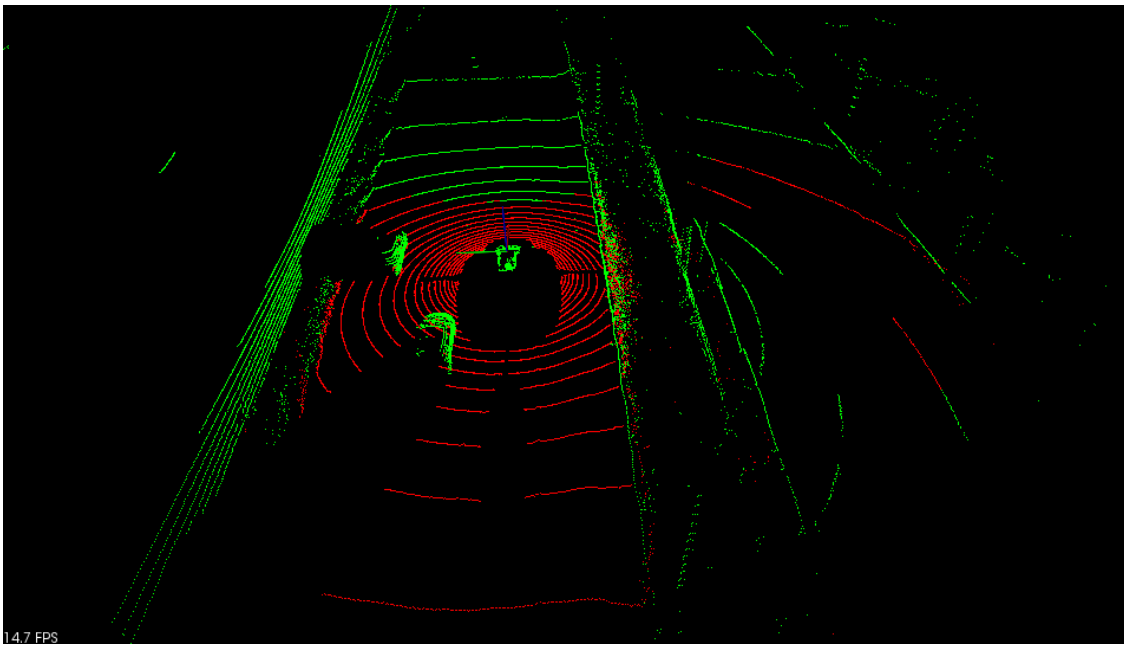


Figura 3.6: Nube de puntos proporcionada por el láser Velodyne (color verde) y puntos *inliers* pertenecientes al plano principal (color rojo).

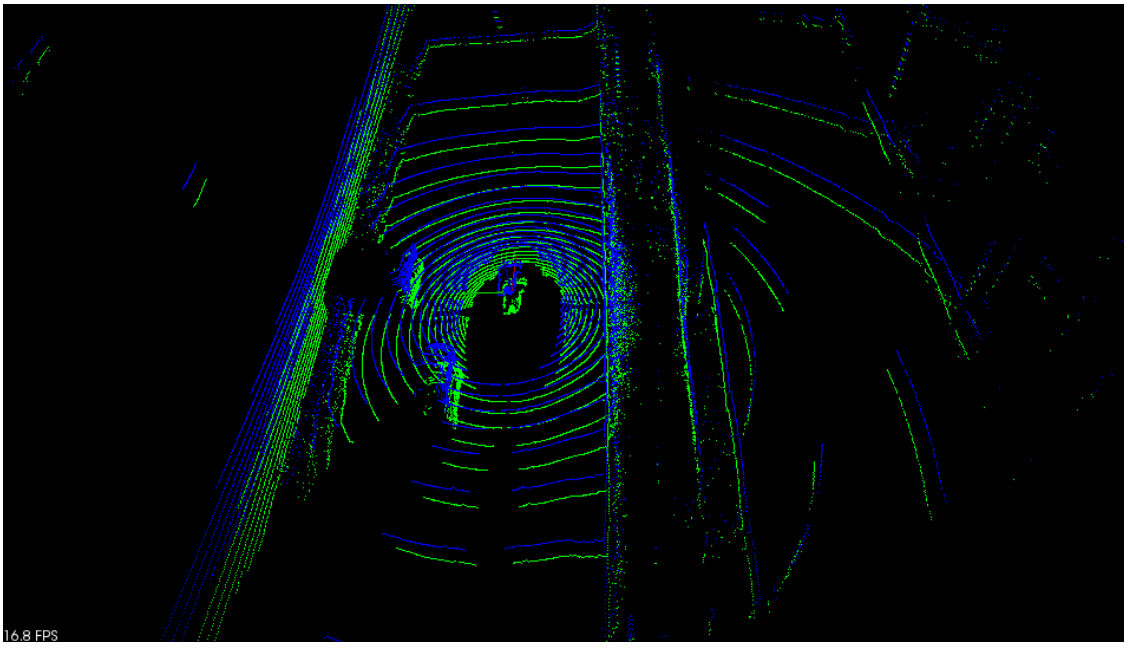


Figura 3.7: Nube de puntos proporcionada por el láser Velodyne (color verde) y nube de puntos ajustada al plano OXY (color azul).

Capítulo 4

Segmentación de la nube para detectar obstáculos

4.1. Detección de obstáculos

El objetivo de la detección de obstáculos es eliminar aquellos puntos de la nube que no cumplan una serie de condiciones para poder etiquetar ese punto cómo obstáculo. Por lo tanto, este algoritmo proporcionará dos nubes de puntos una con los *inliers* (nube de puntos con los obstáculos) y otra nube con los *outliers* (nube de puntos que no contiene obstáculos.

4.1.1. Algoritmo de la tangente

Para seleccionar que puntos de la nube son obstáculos y cuales no se va a comprobar la distancia del punto estudiado con respecto a sus vecinos.

Como se ha comentado en el apartado anterior 3.1. el láser Velodyne HDL-32E utilizado en este trabajo tiene 32 haces que le permiten tener un campo de visión vertical 41,33 grados. Por consiguiente, se han seleccionado grupos de 32 puntos para aplicar el algoritmo de la tangente. El objetivo que se persigue seleccionando grupos de 32 puntos es observar como varía el entorno de forma vertical para una misma posición horizontal del láser.

De cada grupo de 32 puntos, el punto a etiquetar y los vecinos de este punto se seleccionan siguiendo el siguiente array,

$$orden = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, \\ 26, 28, 30, 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31] \quad (4.1)$$

Cada punto de la nube a etiquetar tiene las siguientes coordenadas en el espacio,

$$p_i = (x_i, y_i, z_i) \quad (4.2)$$

donde, i es el índice del array de la ecuación 4.1, cuyo valor inicial es uno.

A continuación, se seleccionan dos vecinos, el vecino anterior y el vecino posterior, los cuales tienen las siguientes coordenadas en el espacio.

$$p_{i-1} = (x_{i-1}, y_{i-1}, z_{i-1}) \tag{4.3}$$

$$p_{i+1} = (x_{i+1}, y_{i+1}, z_{i+1}) \tag{4.4}$$

La distancia euclídea en el plano XY entre el punto a etiquetar y sus vecinos viene determinada por,

$$d_{xyppi-1} = \sqrt{(p_{ix} - p_{ix-1})^2 + \left(p_{iy} - p_{iy-1}\right)^2} \tag{4.5}$$

$$d_{xyppi+1} = \sqrt{(p_{ix} - p_{ix+1})^2 + \left(p_{iy} - p_{iy+1}\right)^2} \tag{4.6}$$

La diferencia de altura en el eje Z entre el punto a etiquetar y sus vecinos se expresa de la siguiente manera,

$$d_{zppi-1} = p_{iz} - p_{i-1z} \tag{4.7}$$

$$d_{zppi+1} = p_{iz} - p_{i+1z} \tag{4.8}$$

La distancia entre el punto a etiquetar y sus vecinos viene determinado por la tangente tal y como podemos observar en la figura 4.1.

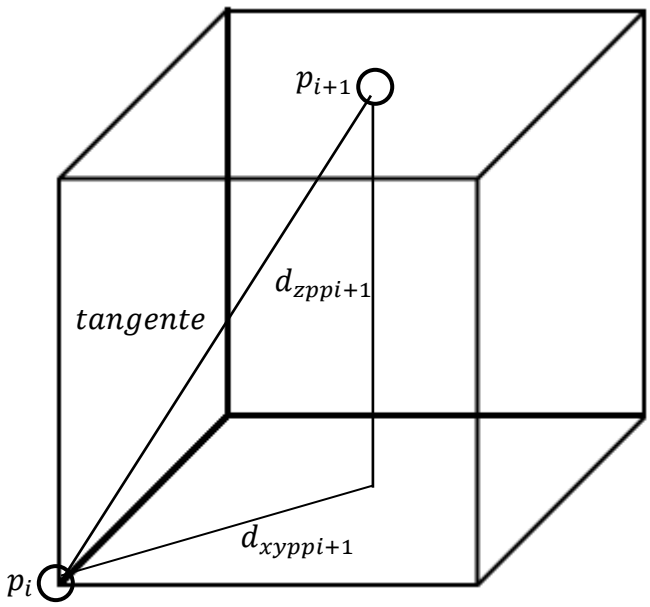


Figura 4.1: Representación algoritmo de la tangente

La tangente se puede expresar mediante la siguiente ecuación,

$$tangente_{ppi-1} = \frac{d_{zppi-1}}{d_{xyppi-1}} \quad (4.9)$$

$$tangente_{ppi+1} = \frac{d_{zppi+1}}{d_{xyppi+1}} \quad (4.10)$$

La siguiente ecuación permite determinar si el punto estudiado pertenece o no a un obstáculo.

$$\left(tangente_{ppi-1} + tangente_{ppi+1}\right) < umbral \quad (4.11)$$

Tras realizar varias pruebas se ha determinado que el umbral mínimo para considerarlo un obstáculo es de 0,76.

Una vez se ha determinado si el punto pertenece o no a un obstáculo se procede a incrementar el índice i hasta que este tome valor 31 y finalice el grupo de 32 puntos de la nube que se ha mencionado anteriormente. Una vez se ha finalizado el estudio de un grupo de 32 puntos, se reinicia el índice para comenzar con el nuevo grupo de 32 puntos.

4.1.2. Condiciones de aplicación del algoritmo de la tangente

Antes de aplicar el algoritmo de la tangente descrito en el apartado 4.1.1. hay que determinar una serie de condiciones en las cuales el algoritmo no se aplicará y, por tanto, los puntos que no cumplan estas condiciones se consideraran *outliers*, es decir, puntos que no son obstáculos.

La primera condición es que los puntos a estudiar deben estar fuera de la zona próxima al vehículo, esta zona está delimitada por un rectángulo tal y cómo se puede observar en la figura 4.2.

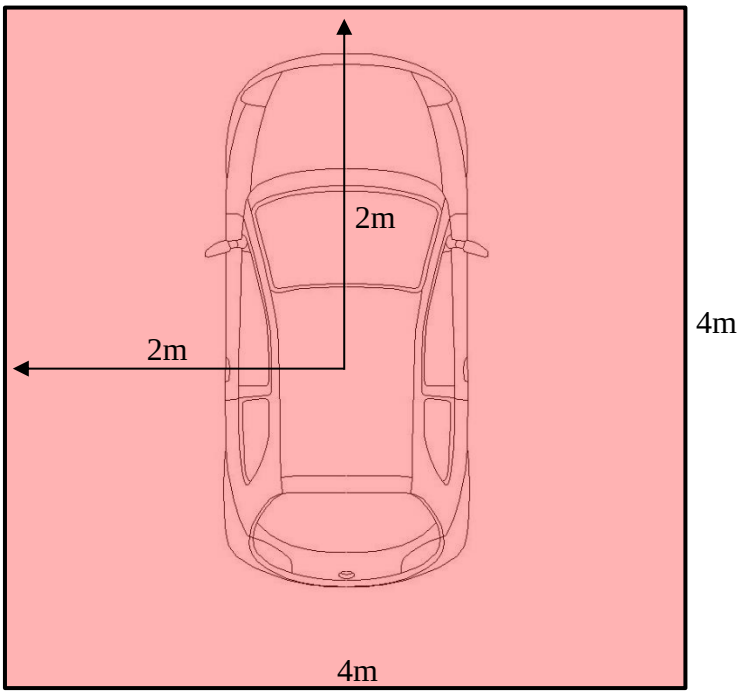


Figura 4.2: Zona próxima al vehículo, los puntos que estén incluidos en esta zona se considerarán *outliers*.

Para aplicar el algoritmo es necesario que se cumplan la siguiente ecuación:

$$(p_i x > 2 \; or \; p_i y > 2) \; and \; (p_{i-1} x > 2 \; or \; p_{i-1} y > 2) \; and \; (p_{i+1} x > 2 \; or \; p_{i+1} y > 2) \tag{4.12}$$

La segunda condición es que los puntos a estudiar cumplan la siguiente ecuación:

$$(p_i z < 4 \; or \; p_i z > -0.1) \; and \; (p_{i-1} z < 4 \; or \; p_{i-1} z > -0{,}1) \; and \; (p_{i+1} z < 4 \; or \; p_{i+1} z > -0{,}1) \tag{4.13}$$

Todos los puntos que no cumplan la ecuación 4.12 o la ecuación 4.13 se considerarán *outliers*. Los puntos que cumplan ambas ecuaciones se determinara su etiquetado mediante el algoritmo de la tangente explicado en el apartado 4.1.1.

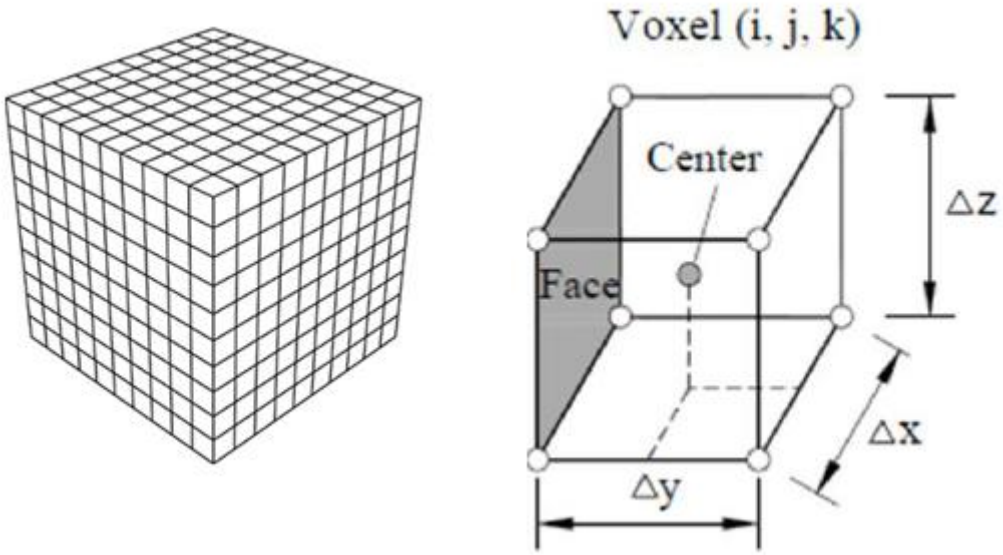
4.2. Extracción de *clusters*

El objetivo de este apartado es obtener un clúster de puntos por cada obstáculo, es decir, una nube independiente por cada obstáculo detectado. El algoritmo partirá con una nube inicial la cual tiene todos los obstáculos, tal y como se ha descrito en el apartado 4.1. y proporcionará una nube independiente por cada obstáculo.

4.2.1. *Voxel grid*

Antes de aplicar el algoritmo se va a realizar un *downsampling* de la nube de puntos inicial, es decir, se va a reducir el número de puntos de la nube. El objetivo de esta reducción de puntos es mejorar el tiempo de cómputo para la extracción de los *clusters.*

El primer paso es dividir el espacio en una rejilla tal y como se puede observar en la figura 4.3.



(a) Espacio dividido en rejillas (b) Rejilla concreta dentro del espacio

Figura 4.3: Espacio dividido en rejillas

Tras varias pruebas realizadas se ha determinado que tamaño óptimo de la rejilla es el siguiente:

$$\Delta x = 0.2\ m \tag{4.14}$$

$$\Delta y = 0.2\ m \tag{4.15}$$

$$\Delta z = 0.2\ m \tag{4.16}$$

Cada *voxel* viene determinado por una longitud en cada uno de sus lados y por su coordenada superior izquierda descrita en la siguiente ecuación:

$$v_{gi} = (x_i, y_i, z_i) \tag{4.17}$$

donde, i es el número del *voxel*.

A continuación, se procederá a determinar que puntos de nube inicial pertenecen a cada *voxel*.

$$p_{nj} = (x_j, y_j, z_j) \tag{4.18}$$

donde, p_{nj} es un punto de la nube inicial y j es el número del punto de la nube.

El punto p_{nj} pertenecerá a un *voxel* si se cumplen las siguientes ecuaciones:

$$v_{gix} < p_{njx} < v_{gix} + \Delta x \tag{4.19}$$

$$v_{giy} < p_{njy} < v_{giy} + \Delta y \tag{4.20}$$

$$v_{giz} < p_{njz} < v_{giz} + \Delta z \tag{4.21}$$

Una vez determinados todos los puntos que pertenecen a un *voxel* estos serán aproximados por un único punto, el cual corresponde con el centroide de todos los puntos de la nube inicial que pertenecen a ese *voxel*, tal y cómo describe la siguiente ecuación:

$$c_{nix} = \frac{\sum_{k=1}^{k} p_{nkx}}{k} \tag{4.22}$$

$$c_{niy} = \frac{\sum_{k=1}^{k} p_{nky}}{k} \tag{4.23}$$

$$c_{niz} = \frac{\sum_{k=1}^{k} p_{nkz}}{k} \tag{4.24}$$

donde, c_{ni} es el centroide del *voxel*, k es el número de puntos de la nube inicial que pertenecen a ese *voxel* y p_{nk} son las coordenadas del punto de la nube inicial.

Finalmente se tendrá una nube de puntos con los centroides de cada *voxel* y, por tanto, se ha conseguido reducir el número de puntos de la nube inicial.

4.2.2. *Kd-tree*

Un kd-tree es una estructura de datos utilizada para organizar cierto número de puntos en un espacio con k dimensiones. Es un árbol de búsqueda binario con restricciones impuestas en él. Estos árboles son útiles para las búsquedas más cercanas de los vecinos. Cada nivel de del kd-tree divide todos los hijos a lo largo de una dimensión específica, usando un hiperplano que es perpendicular al eje correspondiente. En la raíz del árbol, todos los hijos se dividirán en función de la primera dimensión. Cada nivel abajo en el árbol se divide en la siguiente dimensión, volviendo a la primera dimensión una vez que todos los demás se hayan agotado. La manera más eficiente de construir un árbol es usar un método de partición cómo el que usa Quick sort para colocar el punto medio en la raíz y todo con un valor unidimensional más pequeño a la izquierda y más grande a la derecha. A continuación, se repite este procedimiento en los sub-árboles izquierdo y derecho hasta que los últimos árboles que se van a dividir estén compuestos de un único elemento.

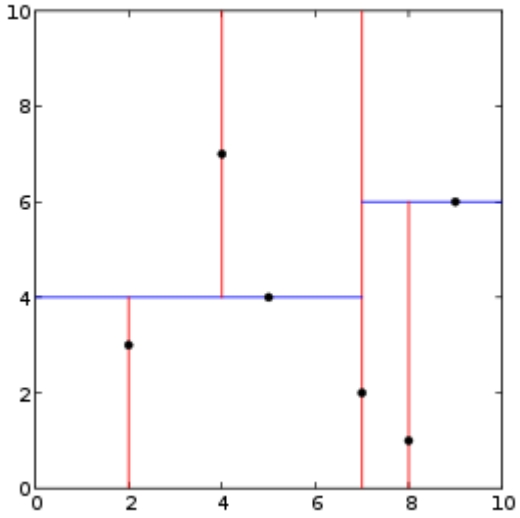


Figura 4.4: Ejemplo de un kd-tree en dos dimensiones

4.2.3. *Euclidean Cluster Extración*

El algoritmo *Euclidean Cluster Extraction* [10] permite obtener *clusters* de puntos que representen obstáculos y así obtener una nube de puntos por cada obstáculo que se encuentre en el entorno.

Un método de agrupación necesita dividir un modelo no organizado de nube puntos P en partes más pequeñas de manera que el tiempo de procesamiento de P se reduzca significativamente. Un enfoque simple de agrupación de datos en sentido euclídeo se puede implementar haciendo uso de una subdivisión de cuadrícula 3D del espacio utilizando cajas de ancho fijo, aunque normalmente, se utilizan estructuras de datos de

tipo *octree*. Sin embargo, en un sentido más general se puede hacer uso de los vecinos más cercanos e implementar una técnica de agrupamiento.

Los pasos de implementación del algoritmo *Euclidean Cluster Extraction* son los siguientes:

1. Crear una representación kd-tree para el conjunto de datos de la nube de la nube de puntos de entrada P, tal y cómo se ha descrito en el apartado anterior 4.2.2.
2. Establecer una lista vacía de *clusters* C, y una cola de puntos que necesitan ser comprobados Q.
3. Entonces para cada punto $p_i \in P$ se realizan los siguientes pasos:
    a. Añadir p_i a la cola actual Q.
    b. Para cada punto $p_i \in Q$ se hace:
        i. Se busca en el conjunto p_k^i de puntos vecinos de p_i en una esfera con radio $r < d_{th}$.
        ii. Para cada vecino $p_k^i \in P_k^i$, se comprueba si el punto ya ha sido procesado, y si no, lo agrega a Q.
    c. Cuando la lista de todos los puntos en Q que se hayan procesado, añadir Q a la lista de grupos C, y reestablecer Q a una lista vacía.
4. El algoritmo termina cuando todos los puntos $p_i \in P$ han sido procesados y ahora forman parte de la lista de *clusters* de puntos C.

4.3. Resultados

En la figura 4.5 se puede observar la nube de puntos rotada tal y cómo se ha detallado en el capítulo 3.

En la figura 4.6 se puede observar la nube de puntos resultante tras aplicar el algoritmo 4.1 descrito en este apartado el cual nos determina los diferentes obstáculos del entorno.

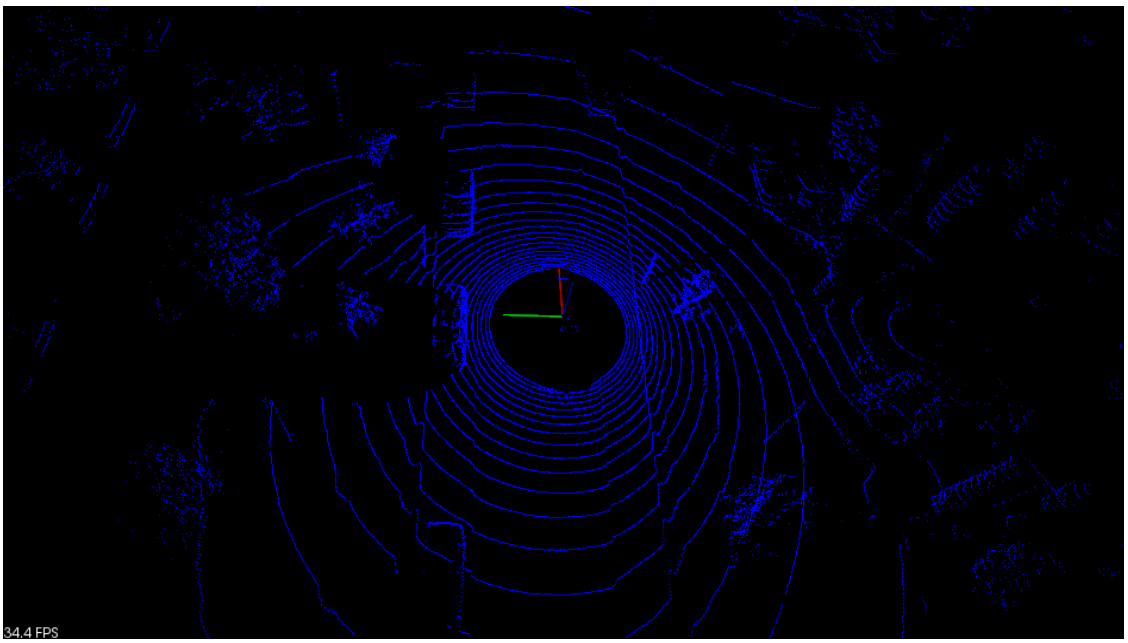


Figura 4.5: Nube de puntos ajustada al plano OXY.

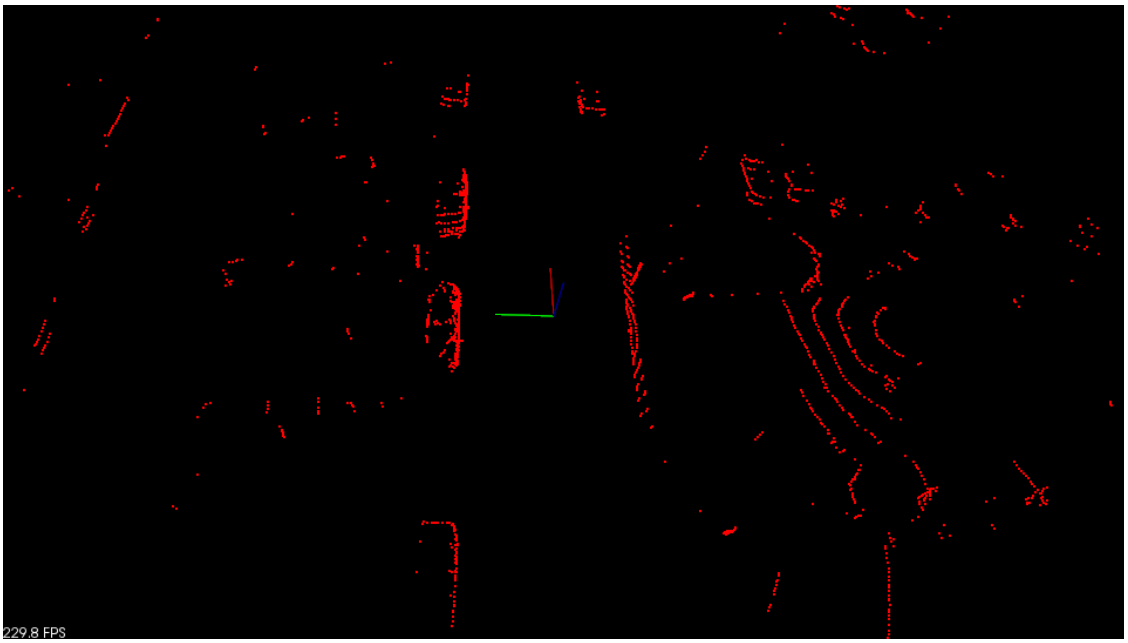


Figura 4.6: Nube de puntos con los obstáculos del entorno.

Capítulo 5

Aplicación de un modelo para determinar ancho, profundidad y ángulo de los vehículos

5.1. Ancho y profundidad de los vehículos

Cada posible obstáculo (*cluster*) determinado en el capítulo 4 está formado por un conjunto de puntos con coordenadas:

$$c_i = (x_j, y_j, z_j) \tag{5.1}$$

donde, i representa el número de *cluster* y j representa los diferentes puntos que pertenecen a ese *cluster*.

El ancho de los vehículos viene determinado por las siguientes ecuaciones:

$$a_{min} = \min(x_j) \tag{5.2}$$

$$a_{max} = \max(x_j) \tag{5.3}$$

$$ancho = a_{max} - a_{min} \tag{5.4}$$

La profundidad de los vehículos viene determinada por las siguientes ecuaciones:

$$p_{min} = \min(y_j) \tag{5.5}$$

$$p_{max} = \max(y_j) \tag{5.6}$$

$$profundidad = p_{max} - p_{min} \tag{5.7}$$

5.2. Ángulo de los vehículos

Para determinar el ángulo de los vehículos con respecto al vehículo equipado con el LiDAR se procederá a aplicar un modelo de recta sobre cada *cluster*. Este modelo buscará en el *cluster* una recta que defina el lateral del vehículo. Esta recta viene determinada mediante la ecuación paramétrica tal y cómo se muestra en la siguiente ecuación:

$$\begin{cases} x = x_1 + kv_1 \\ y = y_1 + kv_2 \\ z = z_1 + kv_3 \end{cases} \tag{5.8}$$

Donde:

- (x, y, z) son las coordenadas de cualquier punto $P(x, y, z)$ de la recta.
- (x_1, y_1, z_1) son las coordenadas de un punto conocido de la recta.
- (v_1, v_2, v_3) son las componentes del vector director de la recta.
- k es un valor real que determina cada coordenada $P(x, y, z)$ dependiendo del valor que se le asigne.

Para obtener las coordenadas de un punto conocido de la recta (x_1, y_1, z_1) y las componentes del vector director de la recta (v_1, v_2, v_3) se ha aplicado el algoritmo RANSAC descrito en el apartado 3.2.1, el cual proporciona los parámetros descritos anteriormente y los puntos de la nube que pertenecen a la recta descrita por esos parámetros (*inliers)* y el resto de puntos de la nube que no pertenecen a esa recta (*outliers).*

El ángulo del vehículo viene determinado por la pendiente de la recta descrita en la ecuación 5.8 en el plano XY. La ecuación paramétrica de la recta en el plano XY está descrita por la siguiente ecuación:

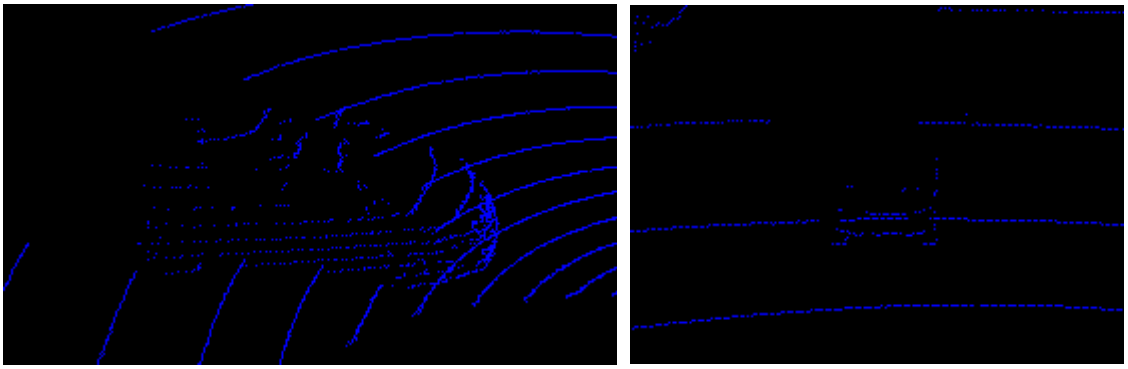
$$\begin{cases} x = x_1 + kv_1 \\ y = y_1 + kv_2 \end{cases} \tag{5.9}$$

Por definición, la pendiente de una recta es la tangente del ángulo que forma la recta con la dirección positiva del eje OX por tanto, cómo el sistema de coordenadas del LiDAR es el que se muestra en la figura 3.2 la pendiente y el ángulo del vehículo vienen expresadas en las siguientes ecuaciones:

$$m = \frac{v_1}{v_2} \tag{5.10}$$

$$\alpha = \tan^{-1}(m) \tag{5.11}$$

Para aplicar el modelo descrito anteriormente se debe de tener en cuenta dos posibles situaciones tal y cómo se puede observar en la figura 5.1.



(a) Vehículo en que se aprecia lateral.

(b) Vehículo en el que sólo se aprecia la parte posterior.

Figura 5.1: Tipos de situación en los que en los que se pueden detectar los vehículos en el entorno

Cuando se esté en la situación (a) de la figura 5.1 en la que se aprecie algún lateral del vehículo el algoritmo descrito anteriormente devolverá correctamente el ángulo del vehículo con respecto al vehículo equipado con el láser, ya que el algoritmo detectará cómo recta del *cluster* la parte lateral del mismo.

En cambio, en la situación (b) de la figura 5.1 en la que sólo se aprecie la parte delantera o la parte trasera del vehículo el algoritmo proporcionará un ángulo próximo a los cero o 180 grados, lo que significaría que el vehículo estaría cruzado en la calzada. Esto es debido a que RANSAC detectará cómo recta la parte posterior del vehículo.

El ángulo devuelto por este algoritmo puede tomar los valores mostrados en la figura 5.2.

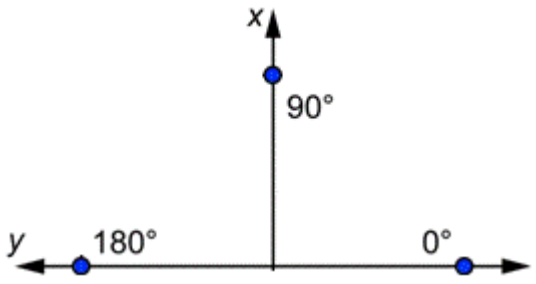


Figura 5.2: Rango de ángulos devueltos por el algoritmo descrito en este apartado.

Con el objetivo limitar el problema descrito anteriormente se tomarán en cuenta las restricciones descritas en la siguiente ecuación:

$$\begin{cases} \alpha, si\ 160^{\circ} > \alpha > 20^{\circ} \\ 90^{\circ}, si\ 160^{\circ} < \alpha < 20^{\circ} \end{cases} \quad (5.12)$$

donde, α es el ángulo devuelto por el algoritmo descrito en este apartado.

Con las restricciones descritas en la ecuación 5.12 se elimina la posibilidad de que un coche estuviera cruzado en la calzada. Esto significaría que se ha detectado la parte anterior o posterior del vehículo y, por tanto, este vehículo está circulando en la misma dirección que el vehículo equipado con el láser.

5.3. Aplicación del modelo

En el capítulo 4 se han extraído *clusters* de los diferentes obstáculos del entorno, pero todos los obstáculos detectados no tienen por qué ser vehículos.

De todos los obstáculos detectados únicamente se aplicará el modelo a los obstáculos que se encuentren dos carriles a la derecha o la izquierda del vehículo que lleva el láser. Por tanto, es necesario que cumplan las siguientes ecuaciones:

$$-7{,}2\ metros < p_{min} < 7{,}2\ metros \quad (5.13)$$

$$-7{,}2\ metros < p_{max} < 7{,}2\ metros \quad (5.14)$$

Para considerar que un obstáculo es vehículo se tienen que cumplir las siguientes restricciones:

$$ancho < 3\ metros \quad (5.15)$$

$$profundidad < 6\ metros \quad (5.16)$$

$$altura < 2\ metros \quad (5.17)$$

Para calcular el ancho y la profundidad de cada uno de los *clusters* se aplicará el algoritmo descrito en el apartado 5.1.

Para calcular la altura de cada *cluster* se tiene que estos están formados por un conjunto de puntos tal y cómo se explica en la ecuación 5.1. El cálculo de la altura se detalla en las siguientes ecuaciones:

$$al_{min} = \min(z_j) \tag{5.18}$$

$$al_{max} = \max(z_j) \tag{5.19}$$

$$altura = al_{max} - al_{min} \tag{5.20}$$

Si no se cumplen las restricciones descritas en las ecuaciones 5.13, 5.14, 5.15, 5.16, 5.17 y 5.20 el *cluster* no se considerará vehículo y por consiguiente no se le aplicará el algoritmo para calcular el ángulo descrito en el apartado 5.2.

5.4. Resultados

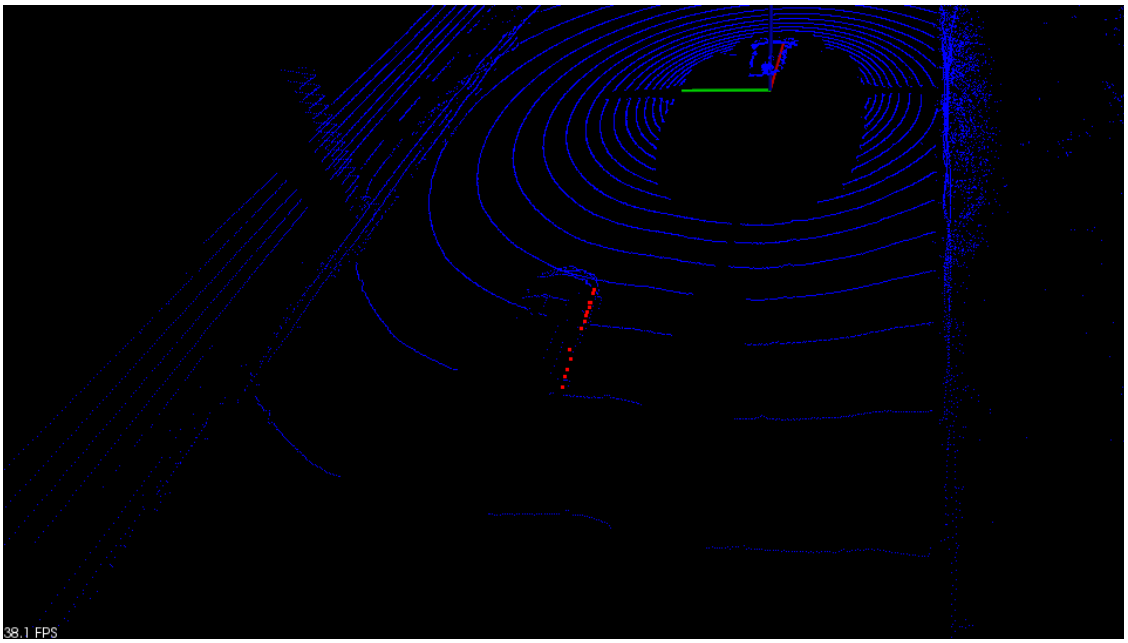


Figura 5.3: Nube de puntos ajustada al plano OXY (color azul) y puntos *inliers* devueltos por el algoritmo RANSAC (color rojo) para el caso de que se detecte el lateral de vehículo.

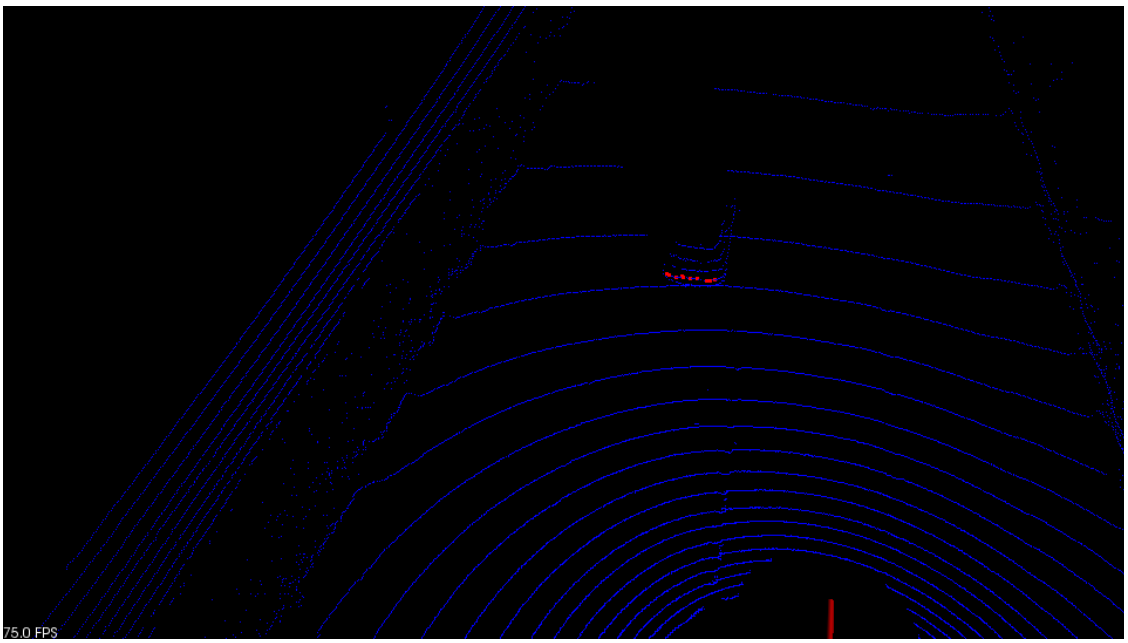


Figura 5.4: Nube de puntos ajustada al plano OXY (color azul) y puntos *inliers* devueltos por el algoritmo RANSAC (color rojo) para el caso de que se detecte la parte posterior de vehículo.

Capítulo 6

Integración temporal de los resultados

6.1. Introducción

El objetivo de la integración temporal de los resultados es realizar un seguimiento entre *frames* de los vehículos detectados, este seguimiento permite mejorar la aplicación del modelo para calcular el ancho, la profundidad y el ángulo de los vehículos. Con la integración temporal de los resultados se obtiene un conocimiento a priori del modelo del vehículo, lo que permitirá mejorar el algoritmo descrito en el capítulo 5.

6.2. Relación de vehículos entre frames

El objetivo es relacionar los vehículos detectados en el *frame* anterior, los cuales ya tienen un modelo de vehículo, con el *frame* actual al que todavía no se le ha aplicado ningún modelo.

Los *clusters* son un conjunto de puntos con coordenadas en el espacio:

$$c_i = (x_j, y_j, z_j) \tag{6.1}$$

$$c_a = (x_k, y_k, z_k) \tag{6.2}$$

donde, i representa el número de *cluster* del *frame* actual, j representa los diferentes puntos que pertenecen al *cluster* del *frame* actual, a representa el número de *cluster* del *frame* anterior y, k representa los diferentes puntos que pertenecen al *cluster* del *frame* anterior.

El primer paso es calcular el centroide de todos los *clusters* del *frame* actual y del *frame* anterior. El centroide del *frame* actual viene determinado por las siguientes ecuaciones:

$$c_{nix} = \frac{\sum_{j=1}^{j} c_{ijx}}{j} \tag{6.3}$$

$$c_{niy} = \frac{\sum_{j=1}^{j} c_{ijy}}{j} \tag{6.4}$$

$$c_{niz} = \frac{\sum_{j=1}^{j} c_{ijz}}{j} \tag{6.5}$$

donde, c_{ni} es el centroide del *cluster* del *frame* actual, j es el número de puntos del que está compuesto el *cluster* del *frame* actual y c_{ij} son las coordenadas en el espacio del punto.

El centroide del *frame* anterior se detalla en las siguientes ecuaciones:

$$c_{nax} = \frac{\sum_{j=1}^{j} c_{akx}}{k} \tag{6.6}$$

$$c_{nay} = \frac{\sum_{j=1}^{j} c_{aky}}{k} \tag{6.7}$$

$$c_{naz} = \frac{\sum_{j=1}^{j} c_{akz}}{k} \tag{6.8}$$

donde, c_{na} es el centroide del *cluster* del *frame* anterior, k es el número de puntos del que está compuesto el *cluster* del *frame* anterior y c_{ak} son las coordenadas en el espacio del punto.

A continuación, se calculará la distancia euclídea en el plano XY entre centroides de cada *cluster* del *frame* actual con todos los centroides de los *clusters* del *frame* anterior. Esto se detalla en la siguiente ecuación:

$$d_{ia} = \sqrt{(c_{nix} - c_{nax})^2 + (c_{niy} - c_{nay})^2} \tag{6.9}$$

Los datos calculados de la distancia euclídea mediante la ecuación 6.9 se organizará en la siguiente matriz:

$$M_r = \begin{bmatrix} d_{11} & d_{12} & \dots & d_{1a} \\ d_{21} & d_{22} & \dots & d_{2a} \\ \vdots & \vdots & \ddots & \vdots \\ d_{i1} & d_{i1} & \dots & d_{ia} \end{bmatrix} \tag{6.10}$$

La matriz de la ecuación anterior 6.10 representa la distancia euclídea de cada centroide del *cluster* del *frame* actual (filas de la matriz) con respecto a todos los centroides del *cluster* del *frame* anterior (columnas de la matriz).

Para determinar que *cluster* del *frame* actual es el mismo que el *cluster* del *frame* anterior se buscará el mínimo valor en la matriz M_r. Con la posición de este valor se tendrá relacionado la fila y la columna y por tanto, el *cluster* anterior y el actual. A continuación, se procederá a eliminar esta posición asignando un valor muy alto a la fila y la columna. Una vez eliminada esa fila y columna se procederá a realizar los descrito anteriormente hasta que se hayan relacionado todas las filas y columnas de la matriz M_r.

Con el algoritmo descrito en este apartado tenemos relacionados los *clusters* del *frame* anterior con los *clusters* del *frame* actual, lo cual nos permite tener un conocimiento del modelo de vehículo del *frame* anterior.

6.3. Aplicación del conocimiento a priori del modelo del vehículo

6.3.1. *Iterative Closest Point (ICP)*

Iterative Closest Point (ICP) es un algoritmo que se emplea para minimizar la diferencia entre dos nubes de puntos. ICP se usa para reconstruir superficies 2D y 3D de diferentes *frames*.

En el algoritmo ICP el *target* se mantiene fijo, mientras en cambio, el *source* se transforma para coincidir de la mejor manera con el *target*. El algoritmo revisa iterativamente la transformación (combinación de translación y rotación) necesaria para minimizar el error, normalmente la distancia desde el *source* hasta el *target*.

Este algoritmo fue introducido por Chen y Medioni [11], Besl y McKay [12].

Esencialmente, los pasos del algoritmo son:

1. Para cada punto de la nube de puntos del *source*, se hace coincidir con el punto más cercano de la nube de puntos del *target*.
2. Se estima la combinación de rotación y translación usando el error medio cuadrático punto por punto usando la técnica de minimización de distancias.
3. Se transforma la nube de puntos del *source* con la matriz de transformación obtenida.
4. Se realizan varias iteraciones.

6.3.2. Aplicación ICP

En el apartado 6.2 se ha explicado como relacionar el *cluster* del *frame* anterior con el *cluster* del *frame* actual, esta relación va a permitir tener un conocimiento a priori del modelo del vehículo, ya que se conoce el modelo de vehículo del *frame* anterior.

El primer paso es aplicar al *cluster* del *frame* anterior c_a el algoritmo de *voxel grid* descrito en el apartado 4.2.1 con el objetivo de reducir el tiempo de cómputo, ya que como veremos a continuación, el *cluster* del *frame* anterior contendrá todos los puntos de los *frames* antiguos, lo que implica trabajar con una gran cantidad de puntos, lo que afecta significativamente al tiempo de cómputo.

Una vez aplicado el algoritmo de *voxel grid* se procede a aplicar el algoritmo de ICP descrito en el apartado 6.3.1. Para aplicar este algoritmo se selecciona como *target* el *cluster* del *frame* actual c_i y como *source* se selecciona el *cluster* del *frame* anterior c_a.

El algoritmo ICP devuelve una matriz de rotación y otra matriz de translación que determina el movimiento de la nube de puntos desde el *source* hasta el *target*, estas matrices vienen determinadas en las siguientes ecuaciones:

$$R = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \tag{6.11}$$

$$T = \begin{bmatrix} T_{11} \\ T_{21} \\ T_{31} \end{bmatrix} \tag{6.12}$$

Una vez se tienen las matrices que relacionan la rotación y la translación desde el *cluster* del *frame* anterior hasta el *cluster* del *frame* actual, se procede a transformar la nube de puntos del *cluster* anterior hasta la nube de puntos del *cluster* del *frame* actual tal y cómo describe la siguiente ecuación:

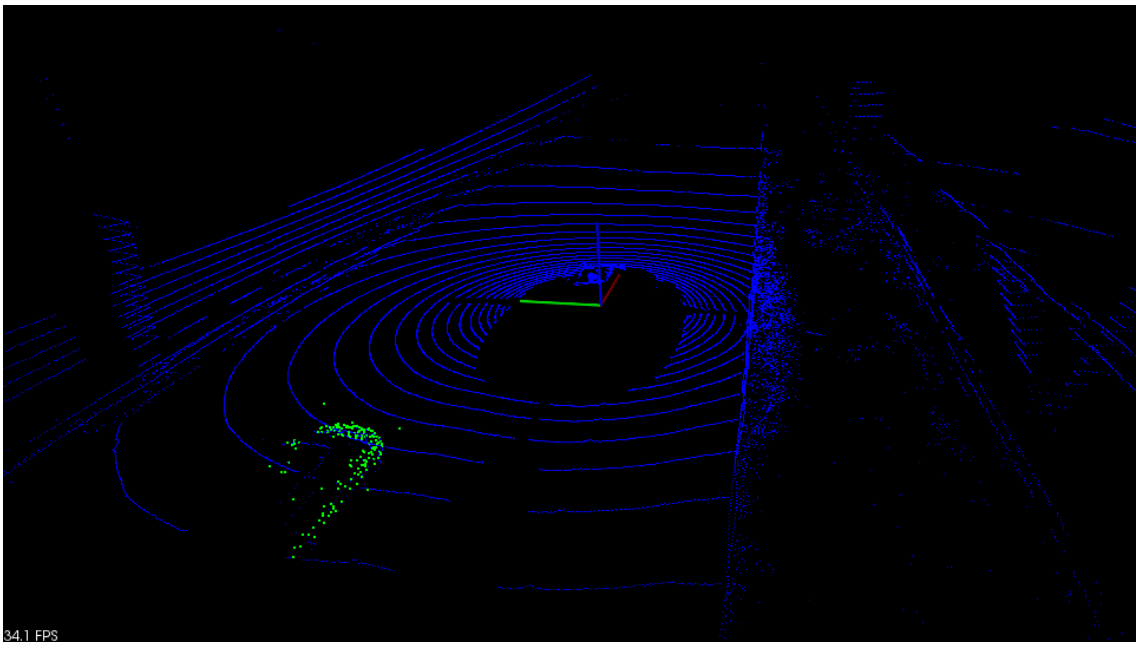
$$\begin{bmatrix} x'_k \\ y'_k \\ z'_k \end{bmatrix} = RT \begin{bmatrix} x_k \\ y_k \\ z_i \end{bmatrix} \tag{6.13}$$

donde, k es cada punto de la nube del *cluster* del *frame* anterior, (x'_k, y'_k, z'_k) es el punto de la nube del *cluster* del *frame* anterior transformado, R es la matriz de rotación 3x3 descrita en la ecuación 6.11, T_p es la matriz de translación 3x1 determinada por la ecuación 6.12 y (x_k, y_k, z_k) es el punto de la nube del *cluster* del *frame* anterior sin transformar.

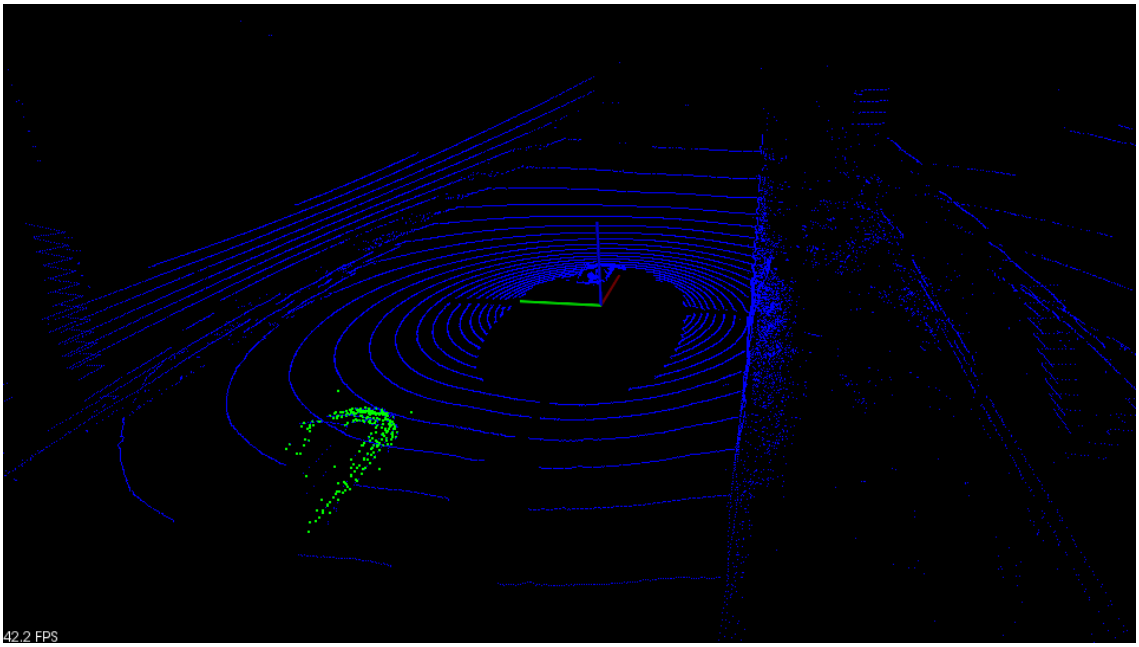
Una vez se tiene el *cluster* del *frame* anterior transformado c_{at} se procede a añadir todos los puntos de este *cluster* al *cluster* del *frame* actual c_i, consiguiendo un *cluster* c_{it} con todos los puntos que ha tenido el vehículo a lo largo de varios *frames*.

Una vez se tiene el cluster c_{it} se procede a realizar los algoritmos descritos en el apartado capítulo 5 descrito anteriormente con el objetivo de conocer el ancho, la profundidad y el ángulo del vehículo.

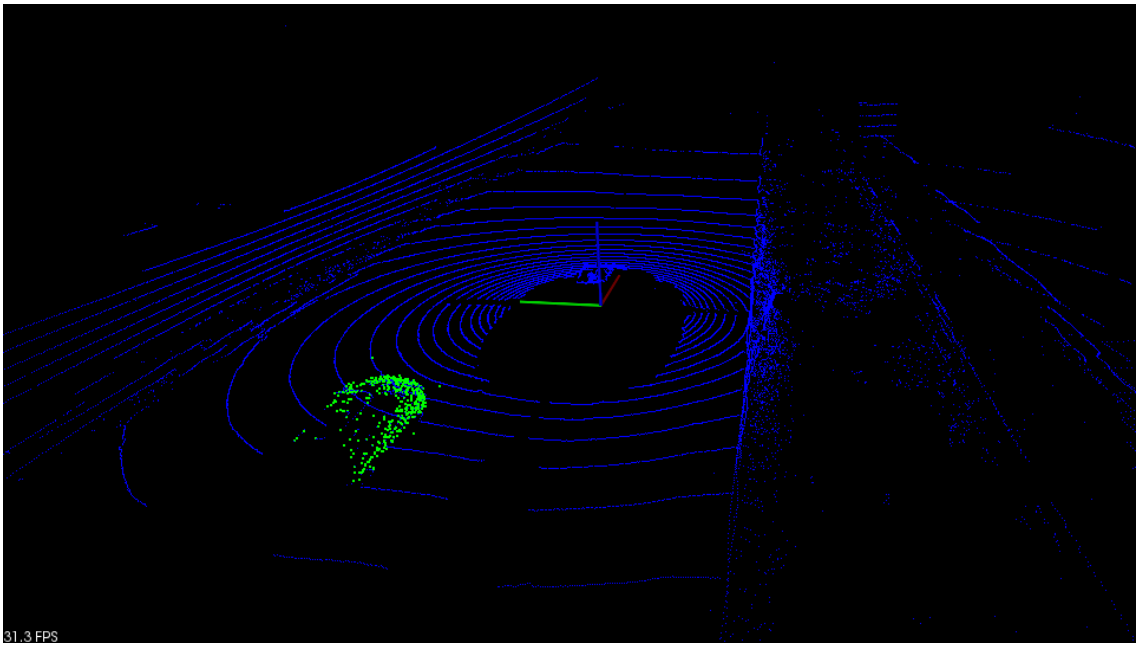
6.4. Resultados



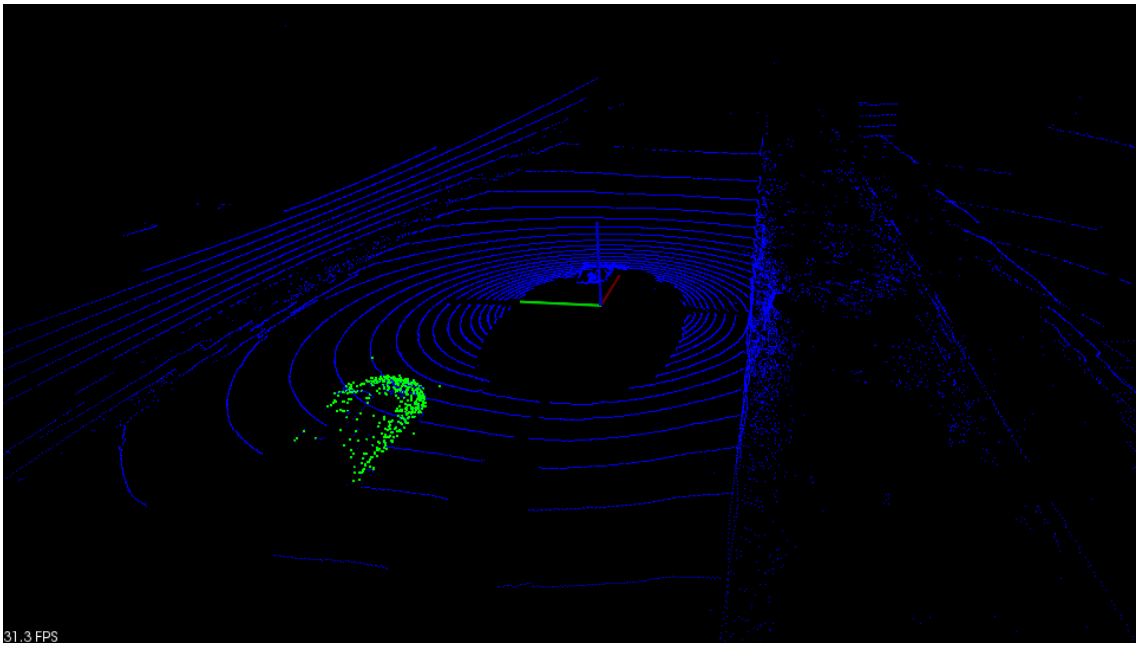
(a) *Frame* 1



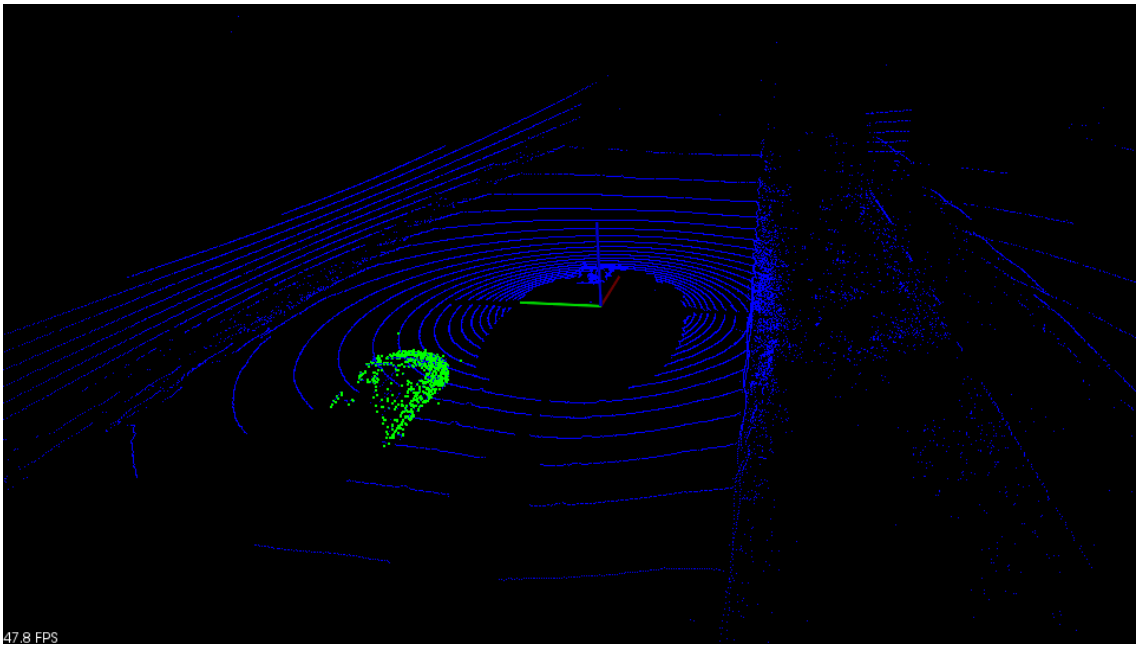
(b) *Frame* 2



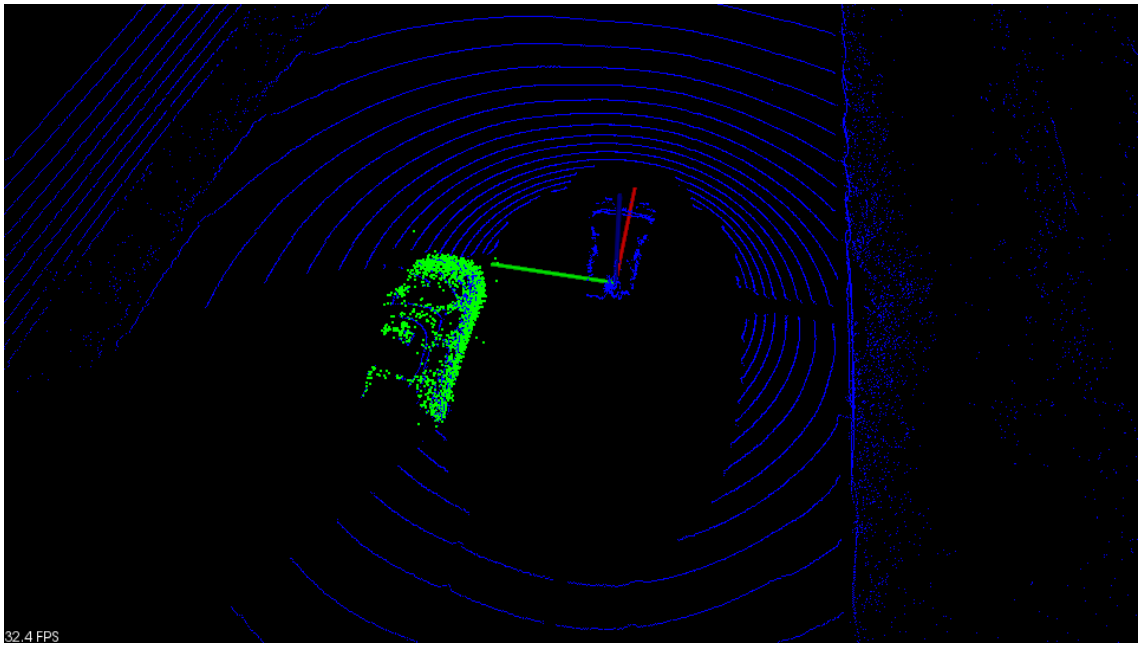
(c) *Frame* 3



(d) *Frame* 4



(e) *Frame* 5



(f) *Frame* 11

Figura 6.1: Nube de puntos generada con el algoritmo ICP descrito en el apartado 6.3.2. a lo largo de diferentes *frames*

Capítulo 7

Resultados

En esta sección se puede observar el funcionamiento del sistema en varias imágenes distintas.

Para marcar la situación de los vehículos en las imágenes se recurre a utilizar el algoritmo descrito en el capítulo 5. Con las ecuaciones 5.4, 5.7 y 5.11, que modelan cada uno de los vehículos, ancho, profundidad y ángulo respectivamente. Se procede a representar un cubo de color blanco en el espacio 3D con los parámetros descritos anteriormente, también, se mostrará un texto en color verde indicando el ángulo del vehículo tal y cómo se puede ver en todos los resultados.

En la figura 7.1 se muestra la situación de los vehículos en una escena urbana. La segmentación del entorno para determinar obstáculos es buena, tal y como se aprecia en la figura 7.1 (b) se distinguen perfectamente todos los vehículos. Por tanto, al aplicar el modelo descrito en el capítulo 5 se obtienen unos resultados correctos. En este caso no se ha aplicado el algoritmo de integración temporal de los resultados descrito en el capítulo 6 ya qué cómo se comentará en las conclusiones el tiempo de cómputo para escenas urbanas es demasiado alto. Pese a esto, el resultado obtenido ha sido satisfactorio.

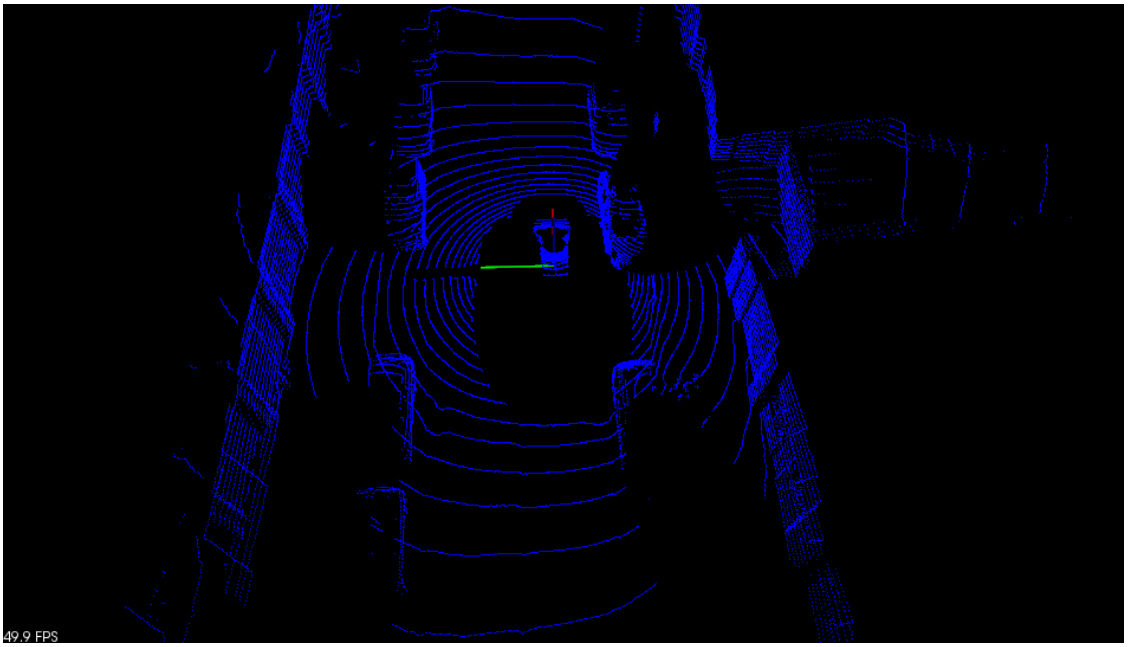
La figura 7.2 muestra una secuencia de una autopista donde se detectan dos vehículos. En esta secuencia se aplica el algoritmo descrito en el capítulo 6 y gracias a este se consigue mejorar los resultados. Esto es debido que se tiene un conocimiento a priori del vehículo permitiendo mantener un ancho y una profundidad más o menos constante a lo largo de los *frames*. El problema de este algoritmo radica en que según avanza el coche el acumulado de puntos del ICP es cada vez mayor y por tanto, afecta significativamente al tiempo de cómputo.

En la figura 7.3 muestra una secuencia donde el sistema no es capaz de realizar correctamente el ajuste mediante el algoritmo ICP. Esto es debido a que la nube de puntos de origen ver figura 7.3 (c) el vehículo situado a la izquierda tiene muy pocos puntos y no se distingue claramente, por tanto, al intentar aplicar el algoritmo ICP se obtiene un resultado erróneo tal y cómo se muestra en la figura 7.3 (d).

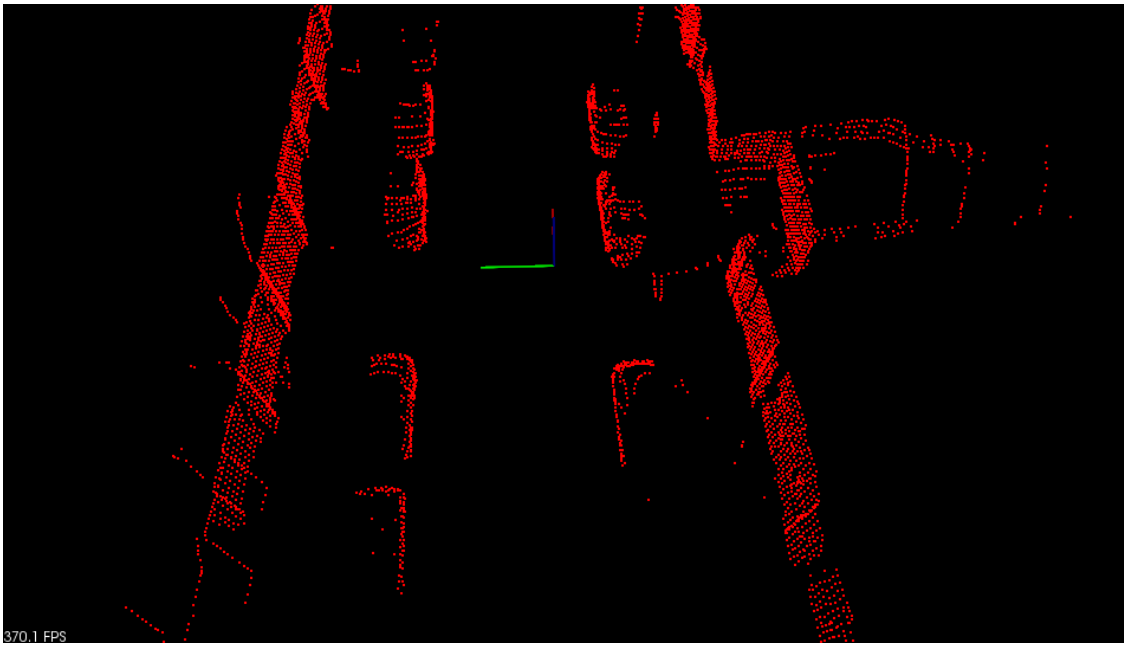
En la figura 7.4 se muestra una situación donde el sistema no es capaz de detectar todos los vehículos existentes en el entorno urbano. En el lado derecho de la calzada se pueden apreciar cinco vehículos aparcados (ver figura 7.4 (a)) en cambio, el sistema sólo detecta cuatro vehículos. El fallo se produce ya que al realizar la segmentación de obstáculos (ver figura 7.4 (b)) y extracción de *clusters*, el *cluster* formado por el vehículo no detectado tiene puntos que no pertenecen al vehículo y, por tanto, al aplicar las

ecuaciones del apartado 5.3 se determina que ese *cluster* no cumple las restricciones para ser un vehículo.

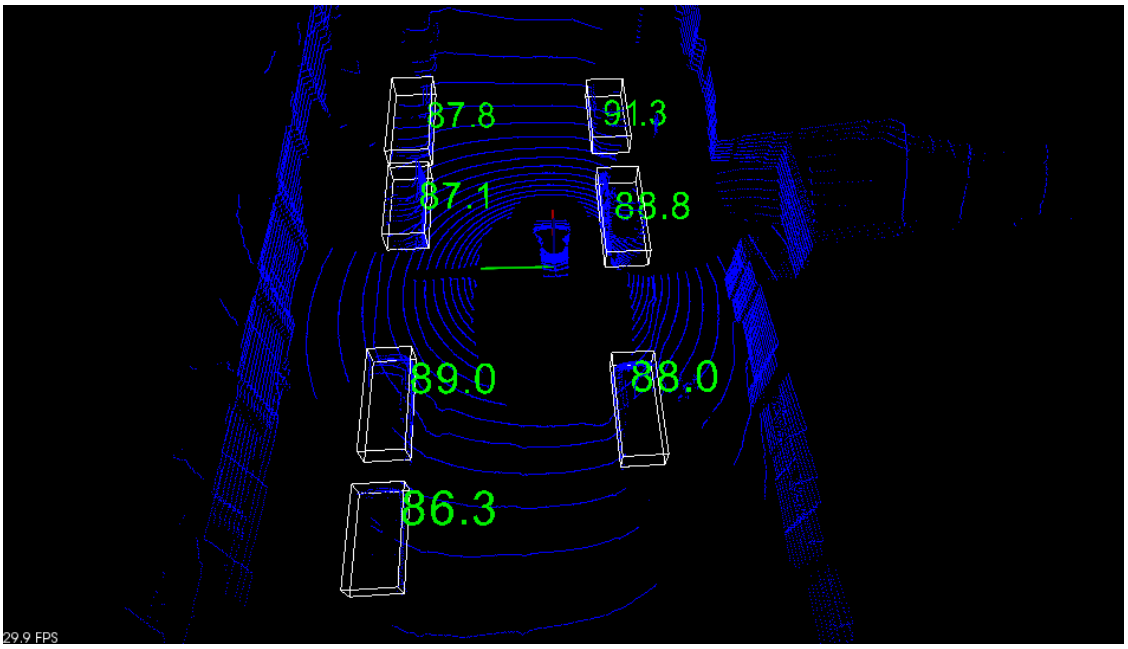
En la figura 7.5 se muestra una situación donde el sistema produce un falso vehículo. En la segmentación de la nube para detectar obstáculos (ver figura 7.5 (b)) se observa cómo hay un obstáculo a la izquierda que cumple las restricciones impuestas en el apartado 5.3 y, por tanto, se considera un vehículo.



(a) Nube de puntos ajustada al plano OXY

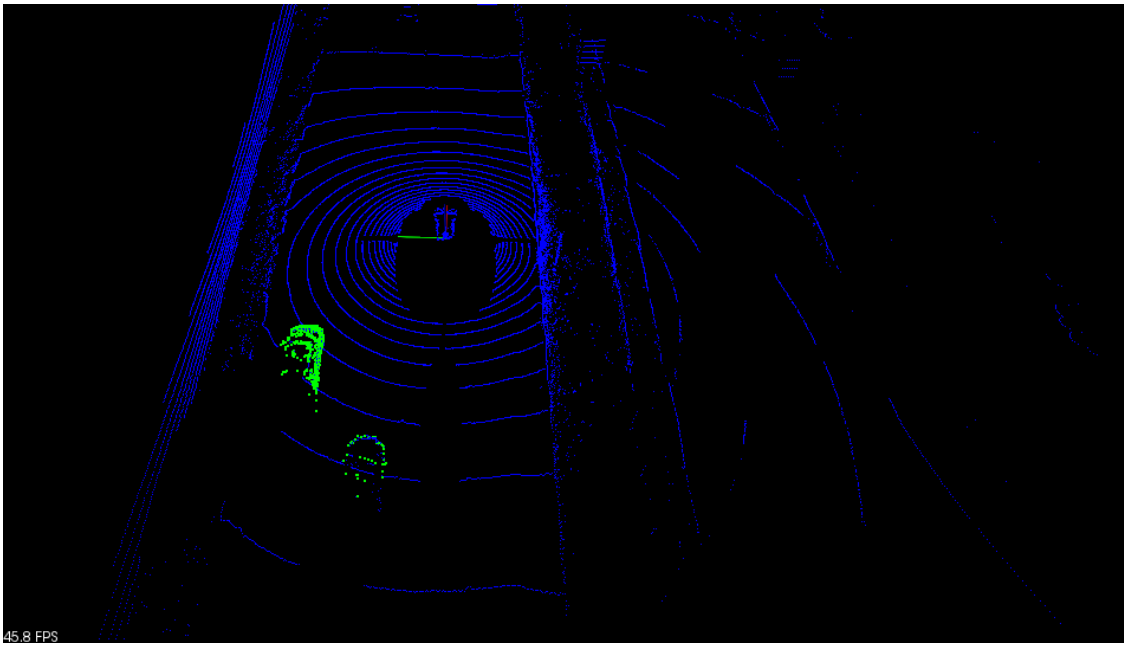


(b) Nube de puntos segmentada con todos los obstáculos.

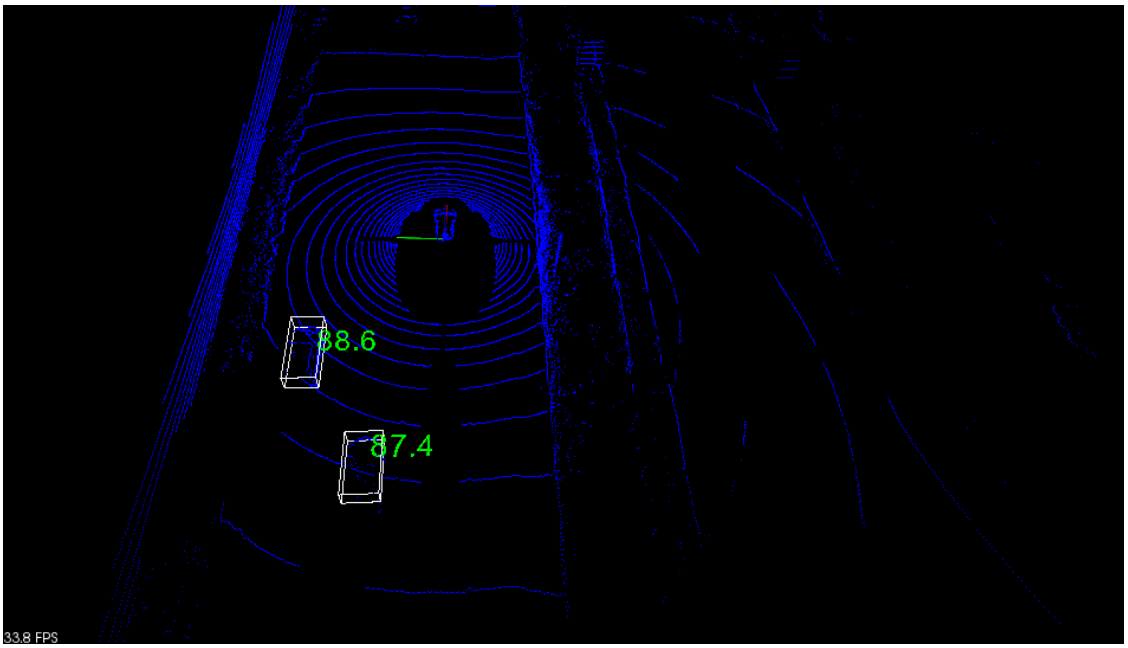


(c) Resultados

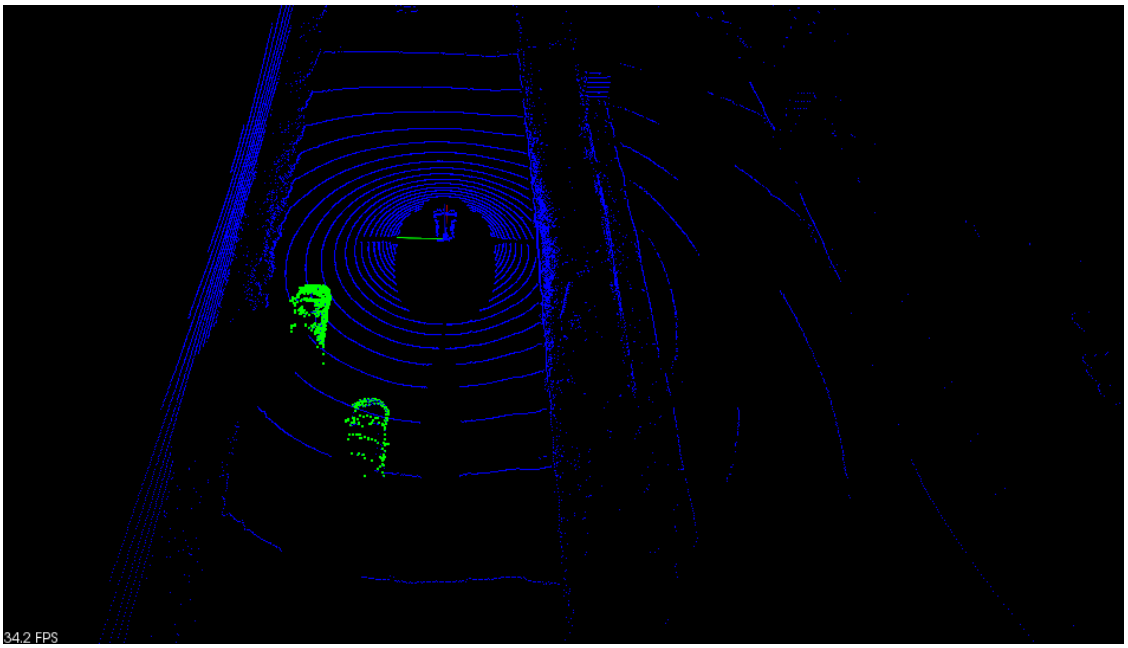
Figura 7.1: Detección de vehículos escena urbana.



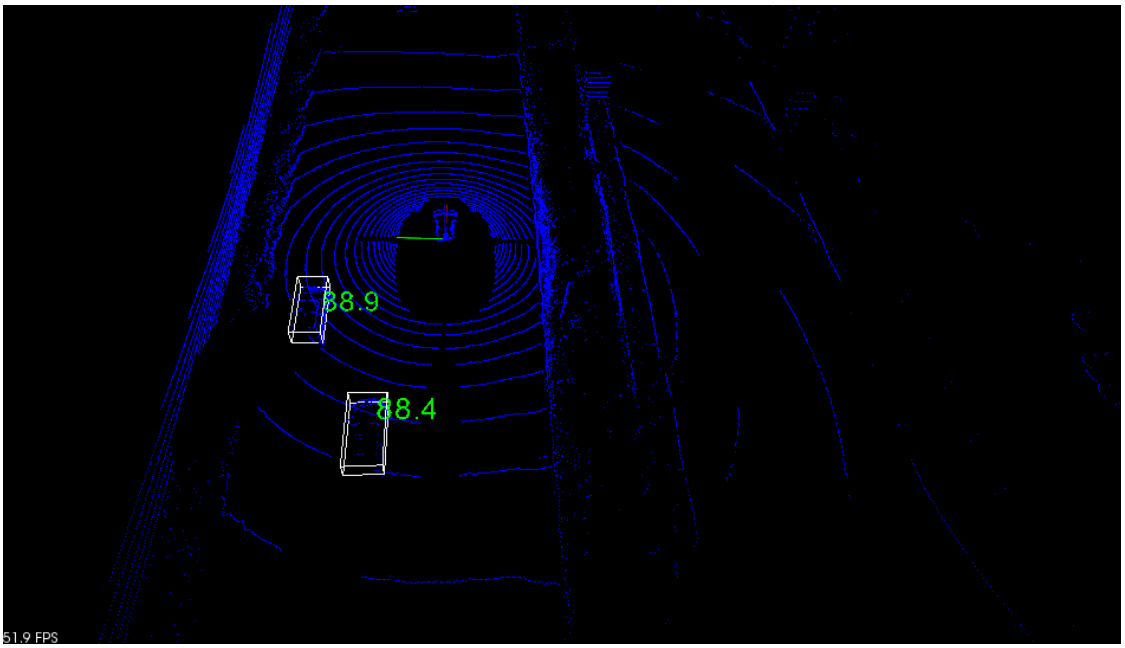
(a) *Frame* 1: Nube de puntos ajustada al plano OXY (color azul) y nube ICP de cada vehículo (color verde).



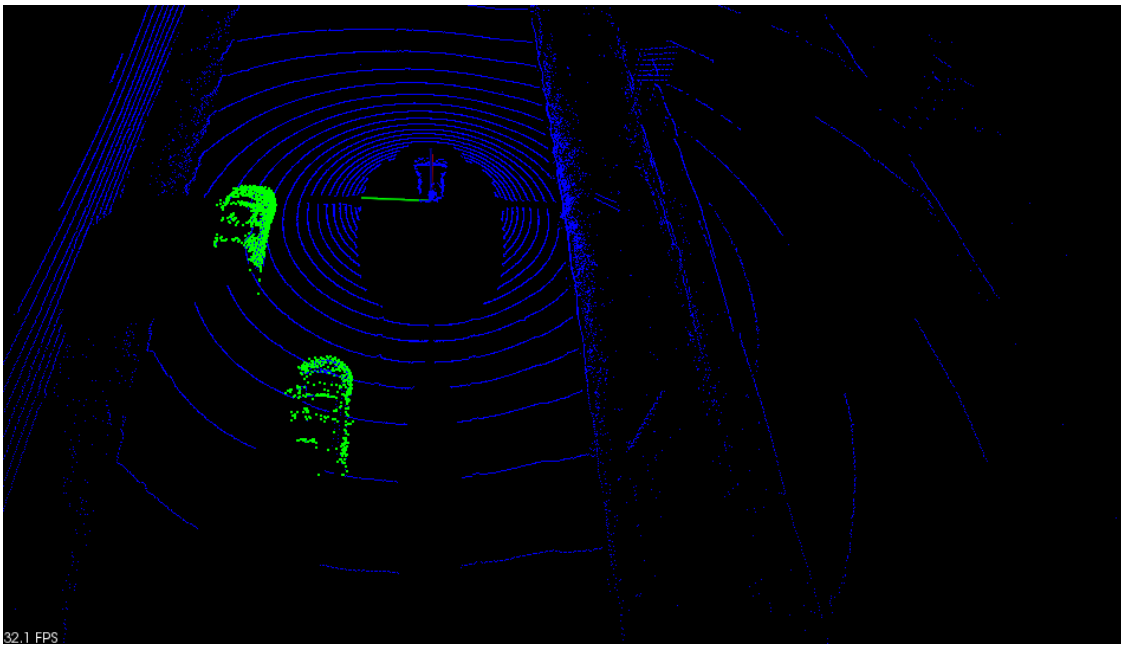
(b) *Frame 1*: Resultados.



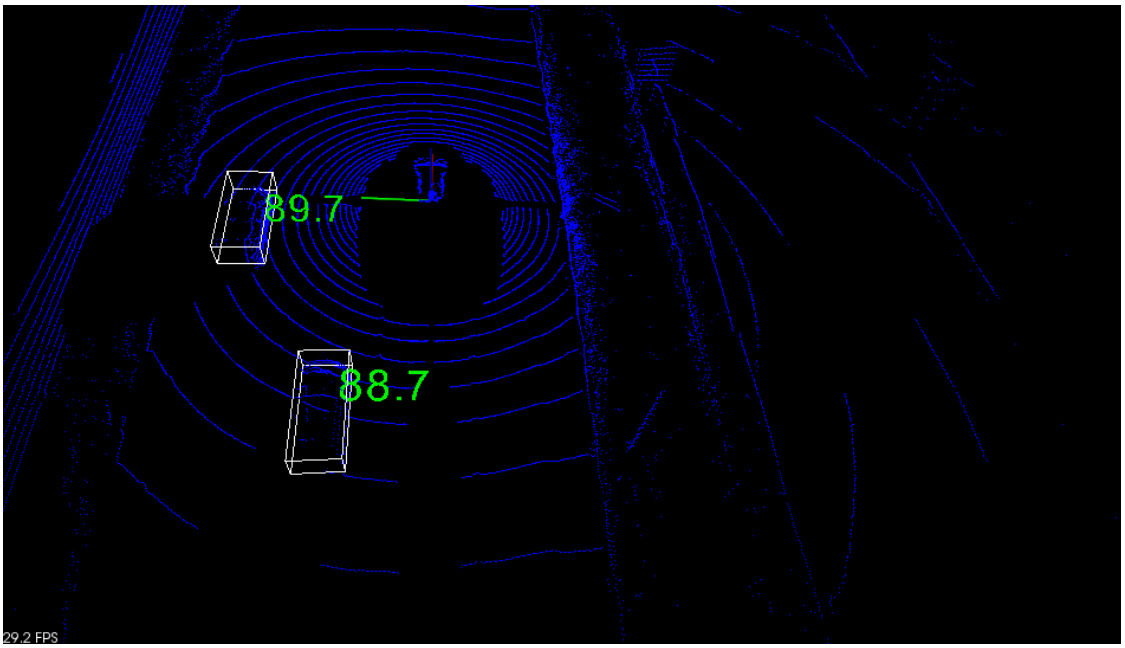
(c) *Frame* 3: Nube de puntos ajustada al plano OXY (color azul) y nube ICP de cada vehículo (color verde).



(d) *Frame 3*: Resultados.

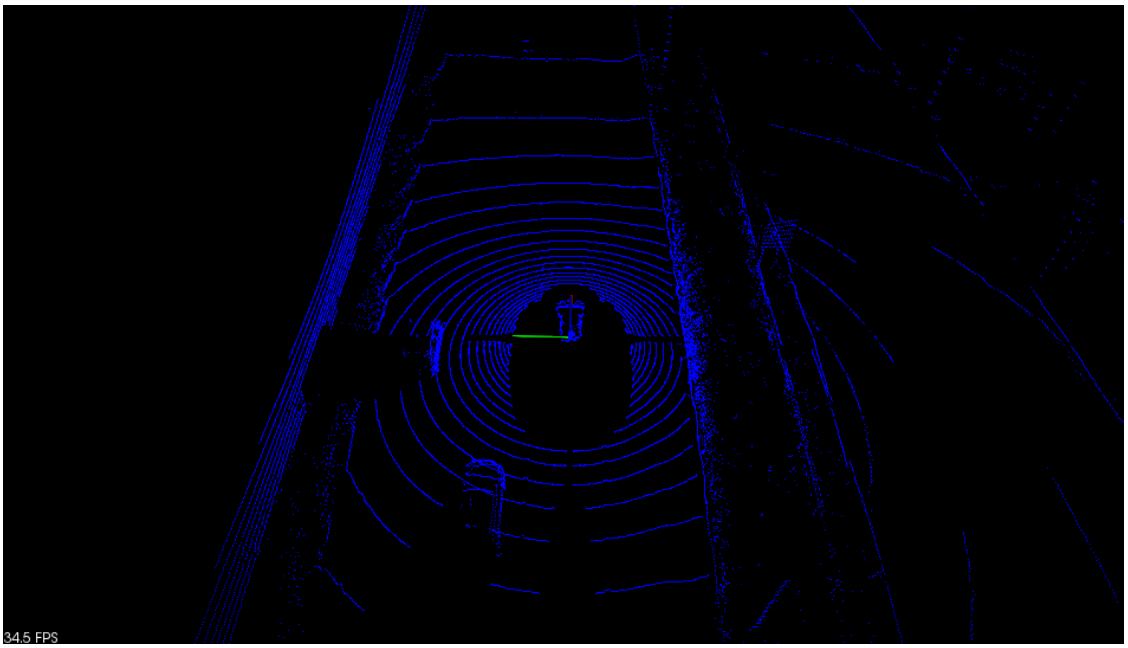


(e) *Frame* 6: Nube de puntos ajustada al plano OXY (color azul) y nube ICP de cada vehículo (color verde).

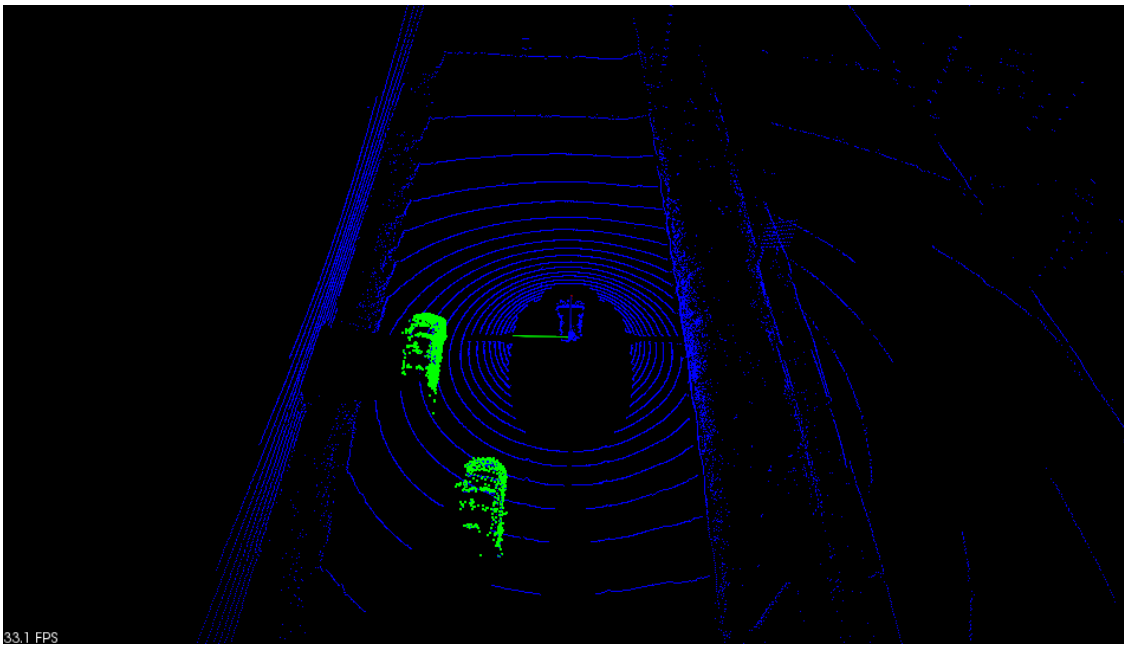


(d) *Frame 6*: Resultados.

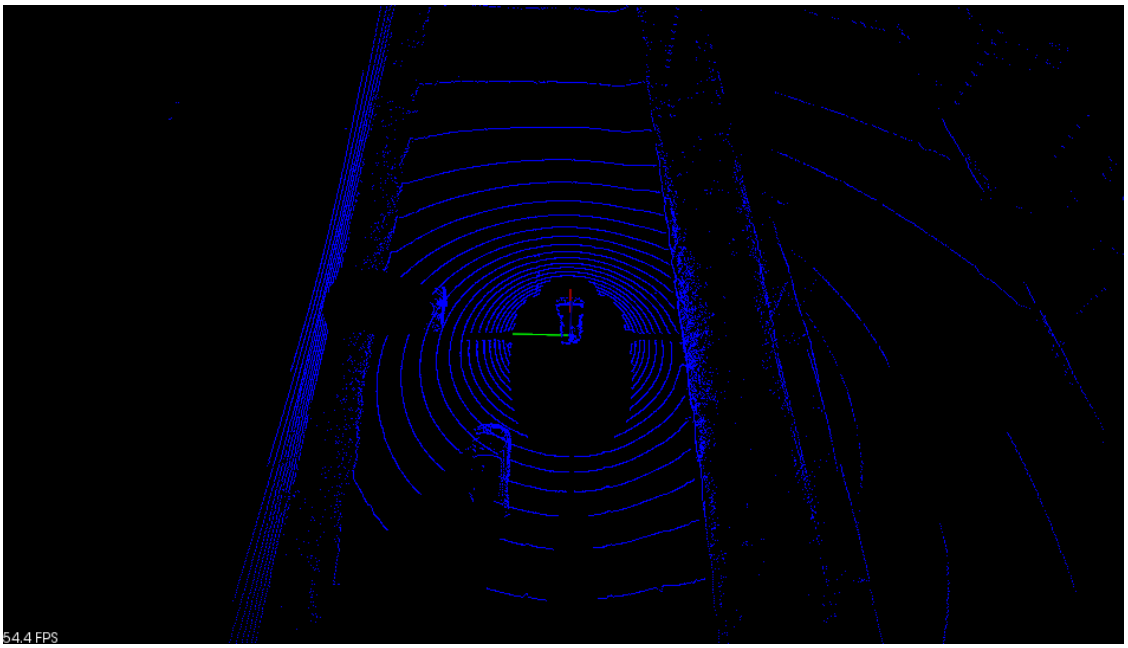
Figura 7.2: Detección de vehículos en autopista utilizando el algoritmo de integración temporal de los resultados.



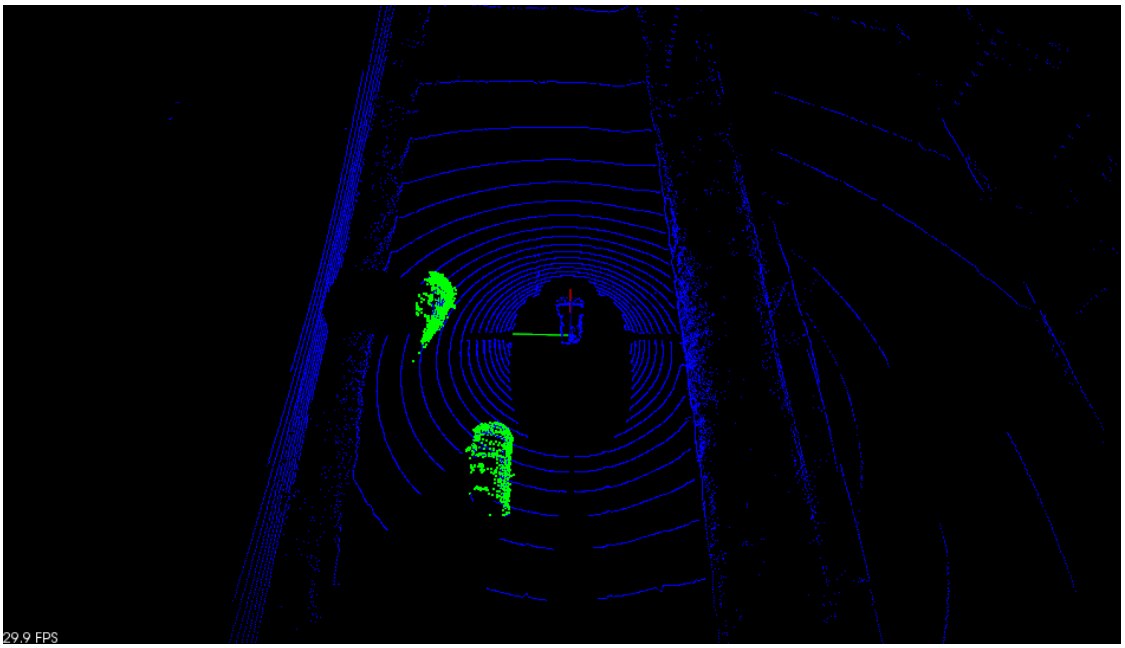
(a) *Frame* 1: Nube de puntos ajustada al plano OXY (color azul).



(b) *Frame 1*: Nube ICP (color verde).

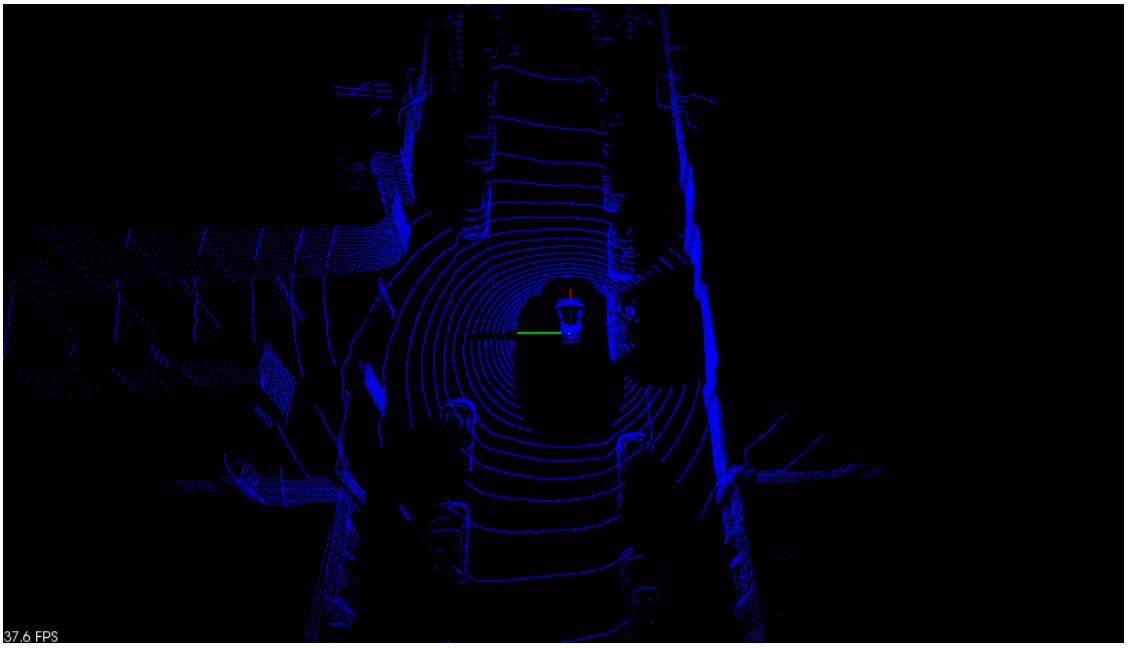


(c) *Frame* 2: Nube de puntos ajustada al plano OXY (color azul).

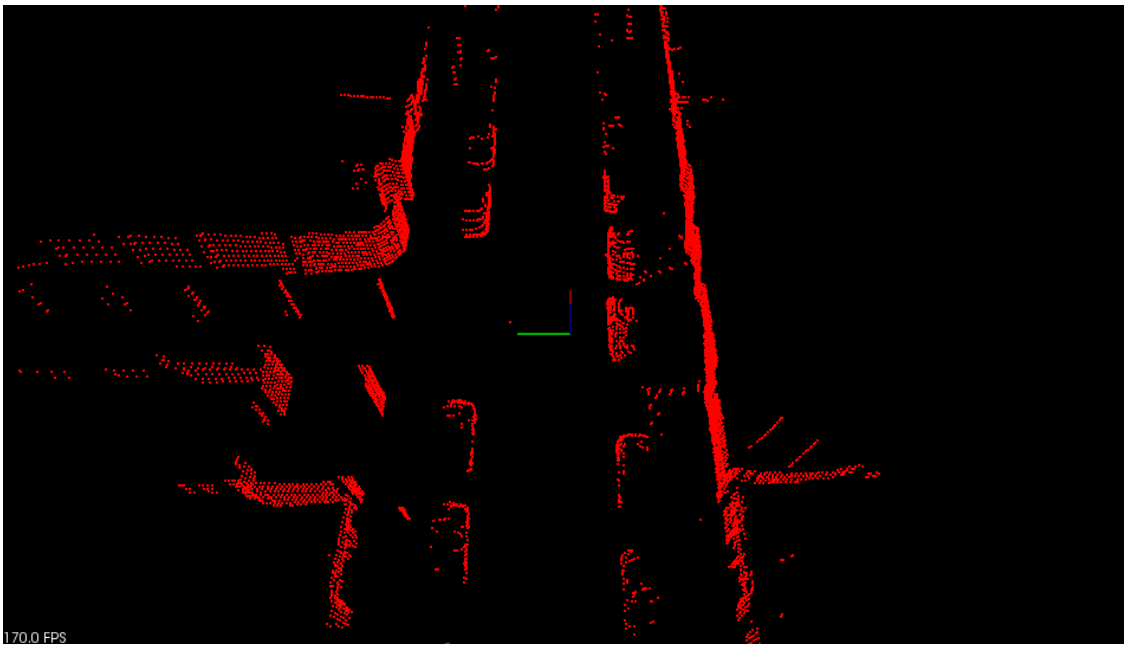


(d) *Frame 2*: Nube ICP (color verde).

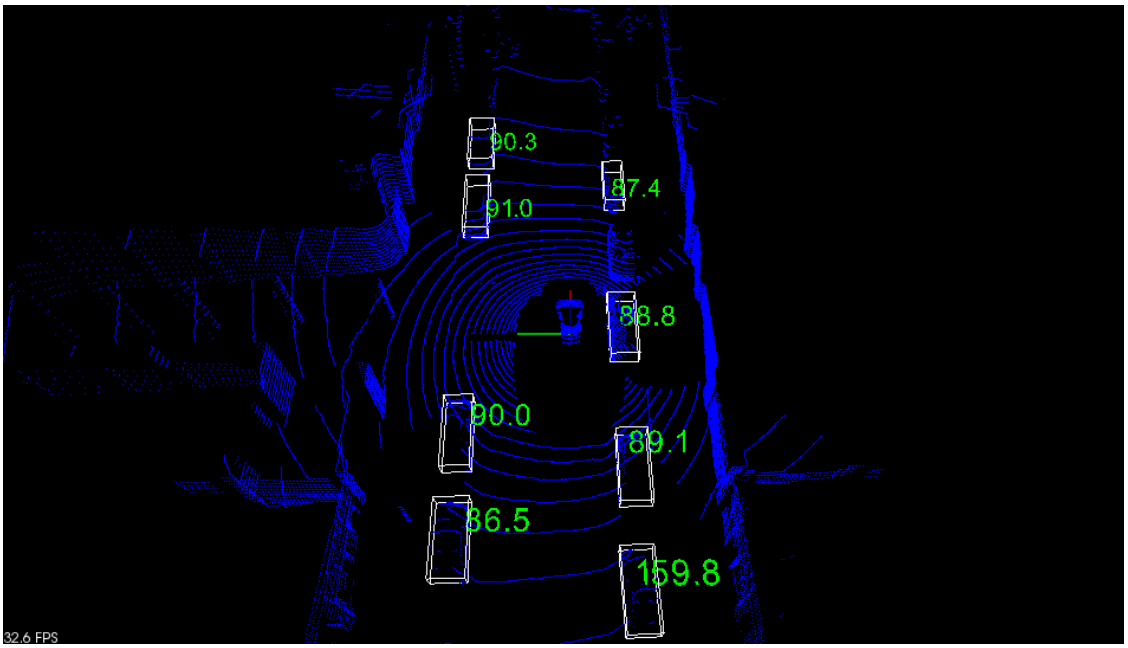
Figura 7.3: Fallo del sistema, error en ajuste del algoritmo ICP.



(a) Nube de puntos ajustada al plano OXY

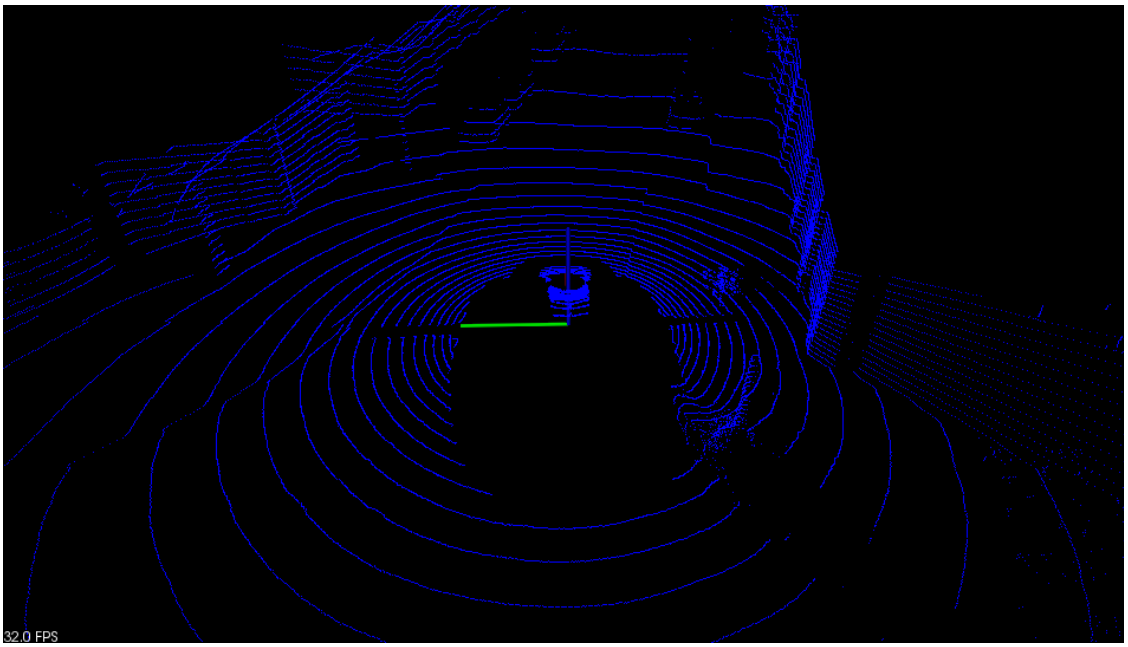


(b) Nube de puntos segmentada con todos los obstáculos.

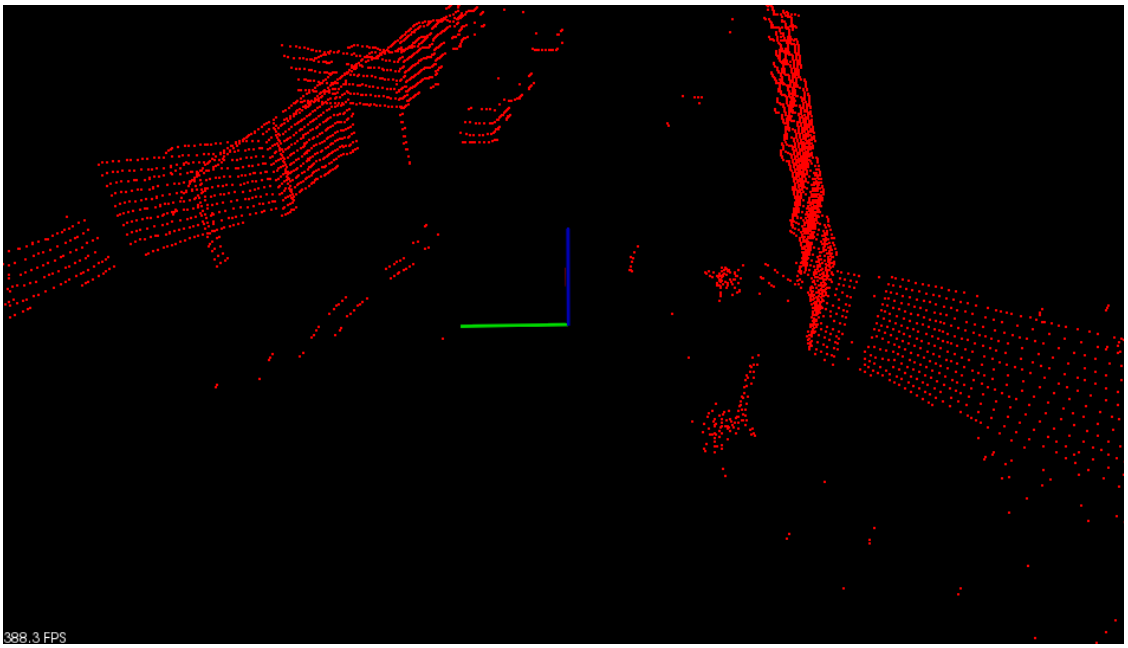


(c) Resultados

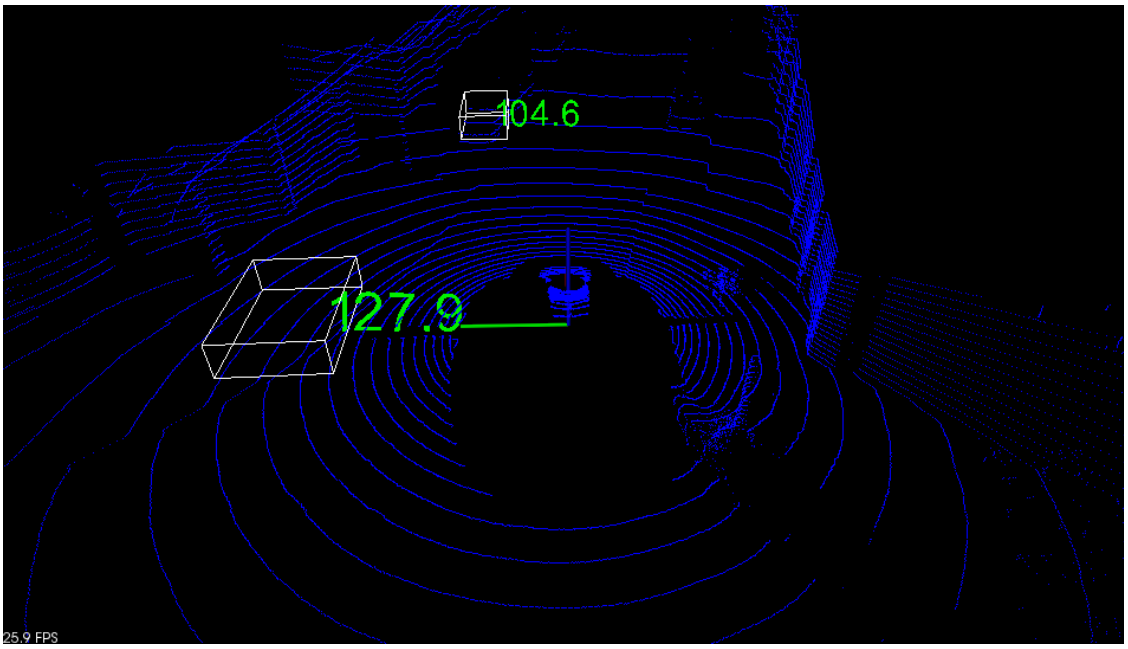
Figura 7.4: Fallo del sistema, no detección de un vehículo.



(a) Nube de puntos ajustada al plano OXY



(b) Nube de puntos segmentada con todos los obstáculos.



(c) Resultados

Figura 7.5: Fallo del sistema, falsa en detección de un vehículo.

Capítulo 8

Conclusiones y trabajos futuros

En este capítulo se exponen las conclusiones globales que se pueden extraer de este trabajo, y las futuras mejoras que se pueden investigar o desarrollar relacionadas con la detección de vehículos.

8.1. Conclusiones

Las conclusiones generales que resumen el proyecto y las principales aportaciones que se han conseguido son:

- Se ha desarrollado un sistema capaz de detectar los vehículos existentes en el entorno de circulación, tanto en ciudad cómo en autopistas y autovías.

- Se han realizado pruebas aplicando en algoritmo de detección de vehículos sin usar la integración temporal de los resultados y las conclusiones son:
    - Los resultados obtenidos han sido satisfactorios.
    - El principal problema es que no se tiene un conocimiento a priori del vehículo y, por tanto, el ancho y la profundidad del mismo varían en cada *frame*.
- Las pruebas realizadas usando la integración temporal de los resultados aportan las siguientes conclusiones:
    - Los resultados obtenidos mejoran a los resultados que se obtienen sin aplicar la integración temporal de los resultados.
    - Unos de los problemas del algoritmo ICP radica en que si el *cluster* del *frame* actual contiene muy pocos puntos el algoritmo no realiza el ajuste correctamente y el sistema falla.
- Los tiempos de cómputos de ambos algoritmos se pueden observar en la siguiente tabla:

	Autopista y autovía		Ciudad	
Método	Promedio (s)	Máximo (s)	Promedio (s)	Máximo (s)
Sin ICP	0,14	0,31	0,19	0,35
Con ICP	0,25	1,35	0,4	1,5

Para realizar el cálculo del tiempo promedio se ha promediado el tiempo de cómputo a lo largo de 2000 *frames*.

Después de observar los tiempos de la tabla anterior se puede determinar que el sistema no funcionaría en tiempo real, ya que el láser proporciona datos cada 100 ms. Sin embargo, el algoritmo se ha probado en un sistema Linux utilizado en una máquina virtual y en un único hilo, por tanto, si este algoritmo se utiliza en un sistema Linux nativo multihilo seguramente se podría aplicar en tiempo real.

8.2. Trabajos futuros

Aunque los objetivos que se propusieron al inicio del TFM se han cumplido, se puede seguir trabajando en otras líneas de investigación y añadir nuevas funcionalidades al algoritmo que mejorarían el sistema de detección de vehículos. Algunos trabajos que se pueden realizar son:

- Realizar mejoras en el *hardware* y optimizar los procesos del algoritmo para reducir considerablemente el tiempo de cómputo y, así, hacerlo viable para su uso en sistemas de tiempo real.

- Proponer otro modelo de vehículo que mejore el cálculo de la profundidad, ancho y ángulo del vehículo.

- Aplicar otra técnica de integración temporal de los resultados que mejore los resultados del algoritmo ICP.

- Realizar un etiquetado de los obstáculos con el objetivo de que los obstáculos que no son vehículos en *frames* anteriores no sea necesario aplicar el modelo de vehículo en el *frame* actual.

Parte III

Bibliografía

Bibliografía

[1] M. Williams, “Prometheus-the european research programme for optimising the road transport system in europe," in *Driver Information, IEE Colloquium on*, Dec 1988, pp. 1/1-1/9.

[2] B. Ulmer, “Vita-an autonomous road vehicle (arv) for collision avoidance in traffic," in *Intelligent Vehicles '92 Symposium., Proceedings of the*, Jun 1992, pp. 36-41.

[3] A. Broggi, M. Bertozzi, A. Fascioli, C. Guarino, L. Bianco, and A. Piazzi, “The argo autonomous vehicle's vision and control systems," *International Journal of Intelligent Control and Systems,* pp. 409-441, 1999.

[4] M. Bertozzi and A. Broggi, “Gold: a parallel real-time stereo vision system for generic obstacle and lane detection," *Image Processing, IEEE Transactions on*, vol. 7, no. 1, pp. 62-81, Jan 1998.

[5] I. Catling, \The drive programme in the european community," in *Driver Information, IEE Colloquium on*, Dec 1988, pp. 2/1-2/4.

[6] I. Catling, R. Harris, and F. Zijderhand, \The socrates projects: Progress towards a paneuropean driver information system," in *Vehicle Navigation and Information Systems Conference, 1993, Proceedings of the IEEE-IEE,* Oct 1993, pp. 319-326.

[7] S. Tsugawa, "Vision-based vehicles in japan: the machine vision systems and driving control systems," *in Industrial Electronics, 1993. Conference Proceedings, ISIE'93 - Budapest, IEEE International Symposium on,* 1993, pp. 278-285.

[8] H. Zhao, C. Wang, W. Yao, F. Davoine, J. Cui and H. Zha, “Omni-directional detection and tracking of on-road vehicles using multiple horizontal laser scanners” in *Intelligent Vehicles Symposium (IV), 2012 IEEE,* June 2012, pp. 57-62.

[9] X. Zhang, W. Xu, C. Dong and J.M. Dolan, “Efficient L-Shape fitting for vehicle detection using laser scanners” in *Intelligent Vehicles Symposium (IV), 2017 IEEE,* June 2017, pp. 54-59.

[10] R.B. Rusu, “Semantic 3D object maps for everyday manipulation in human living environments”, *Institut für Informatik der Technischen Universität München,* 2009.

[12] Y. Chen and G. Medioni, “Object modelling by registration of multiple range images” in *Images Vision Computer Newton, MA, USA: Butterworth-Heinemann,* 1991, pp. 145-155.

[13] P.J. Besl and N.D. McKay “A Method for Registration of 3-D Shapes" *EEEE Trans. on Pattern Analysis and Machine Intelligence*,1992, pp. 239-256.